SYSTEMS FOR TYPE-INDEPENDENT SOURCE CODE EDITING

Inventors:

Britt Worth Piehler
Kevin Zatloukal
David Glen Garber

CERTIFICATE OF MAILING BY "EXPRESS MAIL" UNDER 37 C.F.R. §1.10

"Express Mail" mailing label number: EV 386 446 507 US
Date of Mailing: February 24, 2004

I hereby certify that this correspondence is being deposited with the United States Postal Service, utilizing the "Express Mail Post Office to Addressee" service addressed to **Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313** and mailed on the above Date of Mailing with the above "Express Mail" mailing label number.

[illegible] (Signature)
Tina Galdos
Signature Date: February 24, 2004

SYSTEMS FOR TYPE-INDEPENDENT SOURCE CODE EDITING

Inventors:

Britt Worth Piehler
Kevin Zatloukal
David Glenn Garber

CLAIM TO PRIORITY

[0001] The present application claims the benefit of priority under 35 U.S.C. §119(e) to U.S. Provisional Patent Application entitled "SYSTEMS AND METHODS FOR TYPE INDEPENDENT SOURCE CODE EDITING", Application No. 60/449,984, filed on February 26, 2003, which application is incorporated herein by reference.

COPYRIGHT NOTICE

FIELD OF THE INVENTION

[0003] The present invention relates to the editing of software and software components.

BACKGROUND

[0004] Modern "smart" source code editors provide a wide range of features to the software developer based on increased understanding of the underlying programming language. For example, these editors may provide syntax coloring to highlight various components of the language grammar (class definitions, fields, methods, comments, etc.) The editors may also highlight known errors in the code. In general, this increased understanding of the underlying programming language is achieved by adding a language specific lexical analyzer and/or parser to the editor.

[0005] Unfortunately, most large-scale development projects include several programming languages targeted at different domains and different classes of developers. For example, it is not uncommon for a modern web application to include Java, Java Server Pages (JSP), JavaScript, the Hypertext Markup Language (HTML) and the extensible Markup Language (XML). Therefore, a typical development environment may effectively include several "smart" source code editors, each with an embedded lexical analyzer and/or parser specific to a given language.

[0006] Developing and maintaining a separate editor for each language in the development environment is costly and time consuming. Each time a new language is needed, a new editor must be constructed. Each time a new editing feature is added, it must be added to each language module. In addition, keeping the features of the language editors in sync can be a challenge. Minor differences between the editors in a given development environment can result in an inconsistent and confusing experience for the developer.

[0007] In addition, it is becoming increasingly useful to embed one language inside another within a single source file. For example, JSP pages include Java and JSP tags embedded within HTML. Emerging languages, such as ECMAScript for XML(E4X) embed XML within JavaScript. Other emerging technologies, such as Java for Web Service (JWS) embed a small annotation language inside Java comments to succinctly describe how that Java class should be exposed as a web service. In some cases, several languages can be nested several layers deep in a single source file.

[0008] The simple lexical analyzers and parsers embedded in common source editors are usually not sophisticated enough to recognize and process nested languages. Therefore, in some environments advanced source code editing features are simply not available for nested languages.

[0009] In other environments, a new editor might be constructed specially tailored to handle each new language combination even if separate editors already exist for each of the nested languages. For example, a JSP editor might be constructed to handle the combination of HTML, JSP tags and Java, even though separate HTML and Java editors already exist in the development environment. A new E4X editor may be constructed even though separate ECMAScript and XML editors already exist. This may result in duplication of code and will likely result in inconsistent behaviors as the different language editors evolve.

[0010] As language nesting becomes more popular, the increase in cost and time required to develop and maintain a comprehensive suite of smart editors using traditional methods becomes combinatorial.

[0011] To make matters worse, some nested languages appear in several contexts. For example, XML may be embedded in ECMAScript, Java and JWS annotations. In addition, small expression languages such as those required to understand date and time formats (e.g., YYYY-MM-DDThh:mm:ssTZD from ISO 8601) or time durations (e.g., 15h4m30s) may be embedded in several different languages. Adding these common sub-languages separately to each editor's lexical analyzer and/or parser again results in increased development and maintenance costs and potentially inconsistent behaviors. Any changes to the way these common sub-expressions are handled should be applied uniformly across all applicable host languages.

[0012] In a typical Integrated Development Environment (IDE), there are often two compilers. The first compiler is run from the command line, displaying a list of errors or emitting runable code. The second compiler exists as part of the IDE. Initially, this compiler may only implement lexical analysis of source code in order to support syntax coloring. Then it may implement syntactic analysis in order to support the structure pane and class browser. Eventually, this compiler will contain a nearly complete front-end in order to support code completion.

[0013] The trend of moving more and more of the compiler into the IDE is understandable: advanced IDE features are often based on advanced understanding of the language being edited. Unfortunately, it is not normally possible to use the command-line compiler inside the IDE. First, it is normally not componentized in such a way that the information needed by the IDE is easily accessible. Second, it is usually far too slow for interactive use as changes are made, especially if it takes multiple passes over the files. Third, it almost always recovers poorly from errors, which amongst other problems, makes code completion impossible.

[0014] These issues force the IDE to create its own compiler. However, supporting two compilers has many disadvantages. First, it is nearly twice the work of implementing a single compiler, particularly where the back-end is a fairly high-level language (i.e. Java bytecodes) and no optimization is performed. Second, the IDE's compiler is typically the second class citizen, and as a result, it is usually of lesser quality. Few IDE's actually implement 100% of the analysis in the command line compiler. Furthermore, the IDE's compiler is often designed in an evolutionary manner as new features are needed, resulting in a poorly organized compiler. Third, two different code bases need to be updated in order to make changes to the language. This makes creating a new language a slow and painful process. These problems get worse as the platform is scaled in the number of languages it supports and in the number and sophistication of IDE features.

SUMMARY

[0015] In one embodiment, a source editor capable of editing multiple languages and a compiler framework are configured to communicate with language independent data. In one embodiment, the editor works using compiler meta data that is language independent. Thus, when a new language is introduced into the environment for editing and/or compiling, separate instructions regarding how to integrate the language for compiling or editing are not required. For these and other reasons, the editor provides an edit rich experience without using language specific knowledge.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] FIGURE 1 is an illustration of an editor interface in accordance with one embodiment of the present invention.

[0017] FIGURE 2 is an illustration of an editor interface in accordance with one embodiment of the present invention.

[0018] FIGURE 3 is an illustration of an editor interface in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

[0019] In one embodiment, a source editor capable of editing multiple languages and a compiler framework are configured to communicate with language independent data. In one embodiment, the editor works using compiler meta data that is language independent. Thus, when a new language is introduced into the environment for editing and/or compiling, separate instructions regarding how to integrate the language for compiling or editing are not required. For these and other reasons, the editor provides an edit rich experience without using language specific knowledge.

[0020] To be competitive, a modern IDE should support multiple languages and many sophisticated IDE features. In addition, it is also useful for a compiler to support mixing and nesting languages within the same source file. For example, in emerging language such as E4X, the IDE should display errors for mismatched start and end tags in embedded XML and it should perform auto-completion of XML tags embedded in the source code. These features should be available independent of the host language embedding XML. As another example, JWS annotations should be treated as a nested language and the IDE should support features such as syntax coloring and code completion when editing the annotations.

[0021] Systems and methods in accordance with embodiments of the present invention overcomes problems in existing editing systems by providing and/or utilizing an extensible, data-driven, language independent source code editor with an embedded, extensible multi-language compiler framework. This editor may not include a language specific lexical analyzer or parser. Instead, the editor can be tightly integrated with a compiler framework that provides detailed information about the language currently being edited by the user. This information can be provided in a language-neutral way effectively decoupling the editor from the underlying set of languages being edited.

[0022] In addition, a language-independent editor can expose a set of APIs that makes it easy to customize behavior for specific languages that have characteristics not shared by most languages. This set of APIs can also enable the development of customized views, such as for developing visual editors that represent and allow the user to manipulate aspects of the source code pictorially.

[0023] Multi-Language, Compiler Framework

[0024] A multi-language compiler framework can be used inside the language independent editor. The compiler framework can be used to perform the task of a normal command-line compiler, and can also provide the language information necessary for implementing editor features. Having a single compiler can reduce the amount of work needed to add a new language and to modify and extend that language. It can also ensure that the editor's compiler is of the highest quality.

[0025] In addition, the compiler framework can make it easy to turn language information into editor features. This can allow language designers to focus on their language and not have to worry about implementing the editor-side as well.

[0026] The tight integration of the compiler into the editor, along with the extra time made available by not having to implement a separate compiler for the editor, significantly improves the language-based features of an editor. Here are some examples of the improvements that a compiler framework can make possible:

Performance

The performance many command-line compilers is not good enough for use in an editor, where reparsing occurs each time the user pauses after typing. Therefore, editing a 2000 line file can be very cumbersome.

The compiler framework makes it possible to reparse in “near real-time” with no performance degradation noticeable to the user.

Error Display

In general, editors provide visual indication of errors in a single language.

The compiler framework enables the editor to provide visual indication of errors throughout a source file with mixed languages. Furthermore, the compiler framework keeps track of errors in all source files in the project so that the user can have a complete list of every error in the source, even in unopened files, at all times.

Error Correction

Typical command-line compilers do a poor job of recovering from errors. Often a single error by the user will cause a hundred error messages to display. Poor error recovery also causes other features, like code completion, to be unavailable more often that necessary.

The compiler framework provides parsers that automatically include sophisticated error recovery. This should make most errors cause only a single error message, with the parser continuing as if the error had not occurred. This will make code completion much more robust, with failure a very rare event.

The compiler framework also has error correction in the code-generation of the compiler. This allows the user to run their code even if there are errors in it. Only if the user tries to execute a line for which correction was not possible will it fail.

Auto Correction

The compiler framework also makes it possible to provide the next level of help to users. Instead of just telling the user that there are errors in the code, it can offer to fix them. For example, if the user misspells a variable name, it can provide the user with a list of closely matching names. If the user references a class that is not imported but exists somewhere in the source, it can offer to add an import of the class. If the user forgets a semicolon, it can insert it for them.

[0027] These are only a few examples of the kinds of features a compiler framework can provide by tightly integrating into the editor. The compiler framework can make all of the information produced by the compiler available in real-time for the editor to use, making possible almost any conceivable editor feature.

Compiler Framework Services

[0028] The sections below describe what a compiler framework can do for various consumers of the compiler framework functionality, such as runtime, editor, or language designer consumers. Also included is an exemplary list of languages that can be supported.

Runtime

Produce Annotated .class Files

Pointed at a directory of source code, the compiler framework produces a set of .class files that will correctly implement the semantics described by the source files. Each .class file may additionally contain annotation information (metadata) that also affects runtime behavior. See "Languages" below for the set of supported file types.

Language Designers

Provide Compiler Tools

The framework includes tools to help with building compilers for specific languages. This includes a parser generator. It also includes a scanner generator as well.

Robust to Errors

The generated parsers are able to recover from the majority of user errors (particularly, the common ones). In particular, they are able to recover from all single token errors and also those that occur during code completion (typically, a missing identifier).

The scanners are able to recover sensibly from any error.

Support Language Nesting

The framework allows one language compiler to pass off processing of a section of the document to another language compiler. This language compiler will then get to scan, parse, and type check the contents. The parse tree produced by the inner compiler will be available to the outer compiler.

It is able to choose the language to nest based on type-checking information in the outer language.

It is able to allow either the inner or the outer language to determine where the span of the inner language content ends.

Expose Language Information

The framework allows languages to expose information about the contents of the document in order to enable editor features.

Easy

It is relative easy to expose existing compiler information in order to get editor features. In particular, it should take no more than a few minutes of work to get syntax coloring or bold matching chars from lexical information.

Encapsulates Syntax Details

The exposed information encapsulates the details of the syntax such that the syntax can be changed without breaking the editor features that consume the information.

Editor

Provide Project Information

For the project as a whole, the compiler framework provides the editor with the following information:

1. The names of all classes and packages defined in the source code or libraries.

2. The errors found in any of the source files.

Up-To-Date

All of this information is kept up-to-date for all files in the project. Once the compiler framework is notified of the change to a file, this information should be updated very rapidly.

Provide File Information

For an individual file, the compiler framework provides the editor with the following information:

1. The signatures of the classes defined by the file.

2. The errors found in the file.

3. The stack of nested languages at any point in the file.

4. The information exposed by any of the languages.

Up-To-Date

Once the compiler framework is notified of a change to the file, this information is updated within the time limits for a single-file recompile.

Provide File Information Changes

When the compiler framework is notified of a change to a previously compiled file, it recompiles the file and provides the editor with lists of the changes that occurred to the file information (see "Provide File Information" above).

<u>Languages</u>

Java initially supports the following languages:

Java and JavaScript	The Java and script languages are directly supported. These include files of type .java and .js.
Controls	A set of source code annotations and coding conventions that simplify interaction with external entities, such as web services. These annotations may be embedded in a variety of languages and are defined in control definition files.
JWS	The Java for Web Services (JWS) language for implementing web services includes files of type .jws, .wsdl, and .xmlmap. In addition, it includes support for web services written in script.
JSPX	The JSPX language for implementing web UI includes files of type .jspx and.trd. Additionally, it includes support for web UI written in script.
WebFlow	The webflow language for implementing the flow between pages of web UI includes files of type .jwf (will change). Additionally, it includes support for flow written in script.
WSPL	The WSPL language for implementing business processes management includes files of type .jwf. Additionally, it includes support for processes written in script.

The framework can include direct support for Java with annotations, script with annotations, and XML with Schema. The JWS, JSPX, and WSPL are implemented by extending and nesting the basic languages that the framework provides.

The annotations/schema information can define which tags and attributes are allowed in the document. (In the annotation case, this information may change dynamically based on the Java/script content.) This information can be used to check the validity of the tags and attributes. This information can also include code to perform additional validity checking.

Example Editor Features Enabled Across Languages

[0029] With the rich set of information provided by the compiler framework, it is possible to create a large set of useful source editor features that make it a more powerful tool. Below are some examples.

Editing Features

[0030] The editor for an IDE should know something about the languages it can edit and as a result it can provide a number of useful features which make it easier to edit source files in that language.

Token Coloring

Modern editors provide support for displaying certain tokens, such as keywords, comments and strings, in special colors to help the user better understand the source code.

Comment Editing Help

When editing multiline comments, the editor can insert characters when the user starts a new line. For instance, in Java the user might type "/**" followed by pressing enter, and the editor should insert a "*" automatically, following the standard Java formatting rules for multiline comments (the auto-indenting should also come into play in this situation).

Auto-indenting

When the user is typing certain syntactic constructs, the editor can help them by adding the appropriate indentation when enter is pressed or when certain keys are pressed. For instance, after the user types a "{" and presses enter, the editor can indent the next line by the given indent width. In addition, the editor may automatically indent a line correctly when tab is pressed anywhere on the line, or when the user types certain tokens such as ";".

Matching Tokens

Certain tokens are naturally paired, such as "{" and "}" in Java or C++. The editor may allow the user to move the cursor from one member of a token pair to the other. In addition, it may use a visual indicator to show which tokens are paired either when the token is typed or when the cursor is adjacent to one of these tokens.

Edit by Token

When the user is moving the cursor, selecting text or deleting text, it is frequently useful to be able to do these actions based on token boundaries. For instance, a double-click can be used to select an entire token, control + left/right arrow can be used to move left or right a token at a time.

Code Information

[0031] There are many cases where type information can be used to provide the user with help understanding the meaning of identifiers or to help them understand what function calls and variable references are legal in a certain context.

Completion list

Whenever the user is editing their source code, they should be able to activate a feature which, based on the context in which they are editing, tells them possible text that may be inserted. There are a number of places where this feature could be used:

After the "." on an object.

After the "." on a package (in imports or elsewhere).

After the "new" keyword.

After the "<" on an XML start tag.

After the "</" on an XML end tag.

Parameter Information

When the user is editing the argument list for a method call, the editor may show a list which displays the different legal argument signatures, including the types and argument names (if available). As the user edits the signature, this list displays which argument the user is editing and shows which signature are still legal based on the types of the arguments the user has already entered.

Identifier Information

When the user mouses over (or otherwise selects) an identifier, full information about that identifier can be shown. If it is a variable, the type of the variable can be shown and if it is a function the full signature can be shown. In addition, the user can be taken to the declaration of the member and cycle through the other uses of that member.

Browsing and Navigation

Class Browser

A class browser allows the user to find out what classes are defined in a project, what members and methods the classes contain and the inheritance relationships between the classes. In addition, the user can typically go to any definition or use of a class, member or method.

Navigation Bar

The navigation bar allows the user to see the classes, members and methods defined in the current file and navigate to the location in the file where these items are defined.

Error Detection & Correction

Squiggly Underlines

When the user enters code that contains an error or warnings, these can be detected without a compile and indicated in the source file (like the spelling error squiggly underlines in MS Word). They may be updated in real time as the user types and when the user selects one of these errors, they can see the full error message. In addition, a complete list of these errors for all files can be displayed in the IDE, so that the user never has to recompile the project when they just want an up-to-date list of their errors.

Error Auto-Correct

Certain types of errors such as leaving out an import or misspelling an identifier have obvious auto-correct candidates which can be determined by an IDE integrated compiler. When the user selects these errors, they can be presented with a list of possible correction options which can be automatically inserted into the source code.

Benefits

[0032] The benefits of the language independent editor are numerous. This section lists several examples of benefits that can be obtained using embodiments of the present invention.

Rapid New Language Support

[0033] Adding new languages to a development environment no longer requires the development of a new smart editor. Because the communication between the compiler framework and the source editor is language independent, new languages can be added without a single change to the editor. The compiler framework will provide a rich set of information about the syntax and semantics of each newly added language, immediately enabling a rich set of smart editor features. This drastically reduces the time and effort required to add a new language to a development environment.

Rapid New Editor Features

[0034] Similarly, decoupling the editor from the specific set of compilers means new editor features can be developed once, but will benefit all programming languages plugged into the compiler framework. It is not necessary to add the new feature to a separate editor for each language.

Consistent Editing Experience

[0035] Because there can be a single implementation for all editor features applied to all languages in the compiler framework, the editor can perform uniformly and consistently no matter what language is being edited. Consequently, users who have become accustomed to certain features in one language can use them in another language. The keystrokes and other gestures required to activate and use those features will be the same. The behavior of the editor will be familiar and unsurprising even if the developer is editing a new an unfamiliar language.

Language Nesting

[0036] Because an editor can be language neutral, it can support arbitrarily nested languages. An underlying compiler framework can consult different language modules for each nested portion of the source code and provides information about the syntax and semantics in a language neutral form. The compiler framework can also inform the editor where each language begins and ends within a source file so the editor can apply different user preferences for each language (e.g., the user might like different syntax coloring schemes for different languages).

[0037] One of the benefits of such architecture is that a new language compiler and a new language editor do not have to be developed for each new combination of nested languages. For example, if the compiler framework already has an XML language module and an ECMAScript language module, nesting XML within ECMAScript requires relatively minor modifications to the ECMAScript language module. It is not necessary to create a new language module to enable this functionality and no modifications to the editor are required.

Common Sub-Languages

[0038] The language independent editor can reduce the time and cost of embedding common sub-languages within several host languages. The sub-language can be developed once as an independent language module and nested inside as many other languages as needed. Detailed information about the syntax and semantics of the sub-language need not be added separately to each host language.

[0039] In addition, an editor may not need to know the information provided by the compiler framework about the sub-language is derived from a different language module. Therefore, the sub-language can be added to an arbitrary number of host languages without requiring any modifications to the Editor.

[0040] Changes to the sub-language can be made in place and will be reflected in all host languages. The user experience working with these sub-languages will be consistent regardless of the host language in which they are embedded. All editor features, including syntax coloring, error reporting and statement completion will be uniform and familiar.

Customized Language Features

[0041] APIs exposed by a language independent editor can allow custom language features to be developed easily and quickly. An API can provide default implementations for all the built-in editor features, and can allow extensions to modify or replace existing features or add completely new features. This extensibility can be very useful when the editor does not provide all the desired features or for unusual languages where the existing features need to be customized.

Customized Views

[0042] A language independent editor can also expose APIs that allow third parties to add custom, language editing views to the editor. For example, a workflow programming language might provide a graphical editor for business processes that allows users to create and modify the business processes by dragging and dropping icons on the display. The underlying source code would be modified simultaneously and source code changes could be viewed in a second window while they occur. Alternately, a web service editor might provide a view for graphically understanding and manipulating how the web service interacts with clients and external entities (e.g., other web services). Error! Reference source not found. **Figure 1** shows an example of a visual web service editor.

Data Driven Editor

[0043] As discussed earlier, the features of the language independent editor can be driven by language independent data provided by the compiler framework. This section describes examples of some of the key pieces of information provided by the compiler framework. A complete description of the API that governs the interaction between the compiler and editor is described elsewhere herein.

[0044] One of the important pieces of information that can be provided by a compiler framework is a stream of token nodes. Each token node can identify the start, end and type of a particular token identified by the compiler. The editor can use this information to provide features such as syntax coloring. For example, Error! Reference source not found. **Figure 2** shows a source file highlighting keywords, identifiers, comments, annotations, attributes and attribute values using different color schemes.

[0045] Another important piece of information that can be provided by the compiler framework is a tree of language nodes representing the nested languages in the file. The compiler framework can determine the first language used in a source file by its file extension (e.g., .java, .jws, .jsp, etc.). The host language, based on its syntax, can identify subsequent languages. For example, the JSP language uses the delimiters <% and %> to identify nested sections of Java code. Each language node identifies where the nested language section starts and ends. In addition, it can identify the name of the language (e.g., via com.bea.compiler.ILanguage) and any additional nested language sections inside of it (via a getChildren() method). A compiler can use this feature to e.g. allow users to specify different editor preferences for different languages.

[0046] A compiler framework can also provide information about the entire project, individual files, text buffers, errors in the code, changes to the code and more.

Principal Compiler Framework Components

[0047] Below are descriptions of the principal components of a compiler framework in accordance with one embodiment of the present invention.

Project Compiler

[0048] The Project Compiler contains the list of source directories and the class path. However, the principal data structure maintained by the project compiler is the type cache (part of the java type namespace).

[0049] The type cache contains Java signatures for all of the classes that exist in the project. Some of those classes come from .class files on the class path and the others come from files in one of the source directories. One of the most important jobs of the compiler in the IDE setting is to keep the type cache up to date by watching for changes in the files in the source directories. This task is performed by one of the worker threads in the thread pool (see below).

[0050] The type cache is indexed by file name and by class name. For each file, the entry contains the current list of errors. This means that at any time the IDE can know which files contain errors and can display those errors without opening the file. For each class, the type cache maintains a list of dependencies. A reverse index of dependencies also exists so that the compiler can quickly determine if changes made have broken dependencies in unchanged files.

[0051] Another important benefit of the type cache is improving the performance of type checking. The type cache allows a single file to be compiled without processing any other files. All external information needed to compile the file is contained in the type cache.

[0052] The project compiler (and its contained type cache) is serializable. The IDE will serialize the final state of the compiler to disk when the IDE is closed so that it can display the available classes when the IDE is reopened without parsing any files (other than those changed since closing).

File Compiler

[0053] A file compiler can be used to perform compilation of a single source file. It is designed to perform incremental compilation. Hence, it can maintain data structures containing the result of the previous scan, parse, and, in the case of a non-Java language, translation into Java classes. When changes are made to the file in memory, the next compile can reuse much of the previous results, vastly speeding up the process.

[0054] One of the unique features of this compiler is its built-in support for nesting of languages. The compiler maintains data structures containing information about where language nesting occurs (according to the last parse of the file). This is critical for the editor, which must react differently depending on which language contains the cursor at any given moment.

[0055] The compiler can support the interoperation of different languages. Specifically, any language can call into any other language. This is accomplished by using a common intermediate language. Since the target platform is the Java VM, the clear choice for intermediate language is Java itself. The compiler has a common Java back-end, which is used by all languages for producing byte codes. Each language is able to translate from its parse tree into Java classes. These classes are placed into the type cache to allow other languages to reference them.

[0056] Also important is the framework for language nesting. The outer language is able to determine where the inner language begins. Either the inner or the outer language may determine where the inner language ends. (In the normal case, the outer language will determine this. However, in special cases, the inner language can as well.) The file compiler will remember where the language nesting occurred for reuse on the next parse. Lastly, the outer language may implement a name resolution interface to allow the inner language to resolve references to names defined outside of the nested language.

Thread Pool

[0057] A thread pool can be used in both the IDE and runtime. In the context of the IDE, all parsing needs to be performed on background threads so that the process may be interrupted (if the user starts typing, for example). In the context of the runtime, the thread pool allows compilation of multiple files to be performed in parallel. Compilation should scale linearly to the number of processors. Naturally, all compiler data structures are implemented with appropriate synchronization. They do not assume that the client is accessing the APIs in a single-threaded manner.

Languages

[0058] Language objects can provide the editor with information needed to implement editor features. Language objects contain a method for retrieving different types of information using keys. If that language provides the information, the result of the lookup will be an object implementing a known interface. If not, the result will be null.

[0059] Standard interfaces exist for the type of information needed to implement standard editor features. Features that only exist for one language are implemented with custom interfaces. Standard interfaces also have default (abstract) implementations. Language implementers that want to provide such information only need to implement the abstract methods of the default implementation.

[0060] As an example, one standard interface provides information about matching characters in the token stream. This may be used to implement several features in the editor, such as the bolding of matching characters and the move to matching character keyboard command. To provide this information for a particular language, the language implementer only needs to implement methods describing which tokens match with which other tokens. The code that performs the search will be provided in the default base class.

[0061] These interfaces do not represent editor features directly. Rather they will represent types of information that is used to implement editor features. The code for turning this information into real features will exist in the editor.

[0062] As described above, data structures in the file compiler allow the editor to retrieve the stack of languages in affect a given point in the code. Maintaining languages as objects that can be retrieved in this manner is important because it provides that the same language features are available no matter where that language is used. For example, XML end tag completion should be available whether the user is editing a WSDL file, an XML map nested inside of an annotation in a JWS file, or XML in a script file. This will occur because all situations return the same language object for the XML part of the source.

Compiler Framework Interaction

[0063] When a new file is added to the project or an existing file is modified, the editor can notify the compiler of the change (e.g., via the interface com.bea.compiler.IProject).

The compiler framework and the editor can both have access to the text buffer containing the contents of the file being edited. Each time the user modifies the file, the following exemplary steps can be taken.

1. The user types a character (or otherwise modifies the file)
2. The editor sends a change notification to the compiler framework identifying the changed file, changed text and the type of change (see the interfaces com.bea.compiler.IFileChange and com.bea.compiler.ITextChange)
3. The compiler framework reads and retokenizes the source updating the Token and Node information
4. The compiler framework then enqueues a task for itself to complete the rest of the compilation in a background thread so the user gets immediate feedback and does not detect any visible delay in typing responsiveness while the compiler finishes processing the change.
5. The editor then repaints the screen giving immediate feedback to the user and showing the syntax coloring associated with the new tokenization.
6. Every 250 milliseconds or so, the compiler framework empties the tasks it has enqueued for itself and completes the remaining steps in the background.
7. The compiler framework compiles the changed file(s) in the background. [Note, a change to one file might actually result in several files being recompiled and e.g., new errors being generated for those files. The compiler maintains a type cache that represents dependencies between files enabling it to determine which files must be recompiled based on a given change.]
8. The compiler framework notifies the editor and IDE of changes indicating which files have changed e.g. using the method com.bea.ide.sourceeditor.DefaultSourceDocument.mergeMetadata().
9. The editor reexamines those files and merges the changes with its own internal representation of the parse tree (see section 0). It generates change notifications for each item it identifies that has changed.
10. The editor repaints the screen showing visual representations of the parsing results. For example, newly introduced errors may be highlighted using squiggly red underlines. If the code structure has changed, the change may be reflected in the structure browser.

[0064] It is important to note that the compiler may only complete a small amount of work needed to give immediate feedback to the user while the user is typing. All the larger tasks can be staged for background computation, so as not to disrupt responsiveness to the user.

Parse Tree Merge Algorithm

[0065] To maintain a positive user experience, it can be important for the merge algorithm mentioned above to be very efficient and to identify the minimal number of changes required to synchronize the parse trees maintained by the editor and the compiler framework. Each change notification generated by this algorithm may result in a significant amount of additional work, which could slow the system down. Therefore, naïve comparison algorithms that tend to "get lost" and generate false positives for portions of the file that have not actually changed may not suffice.

[0066] One merge algorithm with acceptable characteristics is presented below. The algorithm is recursive and is initially called passing the root nodes of the destination parse tree and source parse tree as parameters. The trees are constructed of nodes with edges connecting each parent node to its child nodes. Each destination node has a set of properties, which must be updated based on the associated source node.

```
MergeParseTrees(destinationNode, sourceNode)

 1. For each propery p on the destinationNode, set the value of p to the value of the
    property of the sourceNode with the same name as p.
 2. Let numDestinationChildren = the number of child nodes of destinationNode
 3. Let numSourceChildren = the number of child nodes of sourceNode
 4. Let maxComparisons = minimum(numDestinationChildren, numSourceChildren)

 5. // compare children left to right merging them until a match is not found
 6. Let lastLeftMatch = -1
 7. Let childEqual = true
 8. Let i = 0
 9. while (i < maxComparisons and childEqual== true)
      a. Set childEqual to true if destinationNode.child(i) is equal to
         sourceNode.child(i) (i.e., they refer to the same item in the document))
      b. Otherwise set childEqual to false.
      c. If childEqual == true
           i. MergeParseTrees(destinationNode.child(i), sourceNode.child(i))
          ii. lastLeftMatch = i
      d. i = i + 1

10. // if all children have been compared equal, return
```

```
11. if (numDestinationChildren == numSourceChildren) and (lastLeftMatch ==
    numSourceChildren-1)
        a. return

12. // compare  children right to left merging them until a match is not found
13. Let lastRightMatch = maxComparisons
14. childEqual = true
15. i = maxComparisons – 1
16. while (i > lastLeftMatch and childEqual == true)
        a. Set childEqual to true if destinationNode.child(i) is equal to
           sourceNode.child(i) (i.e., they refer to the same item in the document))
        b. Otherwise set childEqual to false.
        c. If childEqual == true
                i. MergeParseTrees(destinationNode.child(i), sourceNode.child(i))
               ii. lastRightMatch = i
        d. i = i – 1

17. Let gap = lastRightMatch – lastLeftMatch – 1
18. Let sourceGap = numSourceChildren – maxComparisons + gap
19. Let destinationGap = numDestinationChildren – maxComparisons + gap

20. // remove deleted nodes
21. if (sourceGap == 0 and destinationGap > 0)
        a. for j = 0 to destinationGap
                i. destinationNode.removeChild(lastLeftMatch + 1)

22. // add inserted nodes
23. else if (sourceGap > 0 and destinationGap == 0)
        a. for j = 0 to sourceGap
                i. Let child = sourceNode.child(lastLeftMatch + j + 1)
               ii. destinationNode.insertChild(lastLeftMatch + j + 1, child)

24. // same number of nodes in gap. Replace or merge
25. else if (sourceGap == destinationGap)
        a. for j = 0 to destinationGap
                i. Let sourceChild = sourceNode.child(j + lastLeftMatch + 1)
               ii. Let destChild = destinationNode.child(j + lastLeftMatch + 1)
              iii. If sourceChild and destChild are the same type of node,
                   MergeParseTrees(destchild, sourceChild)
               iv. Otherwise, replace destChild with sourceChild in sourceNode

26. // different number of nodes in gap. Remove and Insert
27. else
        a. for j = destinationGap-1 downto 0
                i. destinationNode.removeChild(lastLeftMatch + j + 1)
        b. for j = 0 to sourceGap
```

```
            i. Let $child = sourceNode$.child($lastLeftMatch + j + 1$)
            ii. $destinationNode$.insertChild($lastLeftMatch + j + 1$, $child$)
    28. return
```

Language Nesting

[0067] Because an editor can be language independent, nested languages can be handled. All detailed knowledge about the various languages can be embedded in language modules plugged into the underlying compiler framework. A compiler framework can use language neutral APIs described elsewhere herein to communicate understanding of the language concepts to the editor (e.g., positions and types of tokens, errors, etc.).

[0068] The editor can use the information provided by the compiler framework to determine which language is currently being edited and detect when the user moves the cursor from one language to another. This is useful e.g. if the user wants to establish different editing or display preferences for each language. For example, **Figure 3** shows how different syntax coloring schemes might be applied for the Java, HTML and JSP tag languages in a JSP file.

[0069] The compiler can expose information about the languages used in a source file as a tree of language nodes. Each language node can identify a section of the file written in a particular language. The start position, stop position and information about the language (e.g., its name) are provided. If necessary, the editor can navigate this tree to understand all the languages used in a given source file and how they are nested inside one another.

[0070] The compiler framework can determine the initial language of each file using the file type (e.g., determined by a filename extension). It can then pass the file to language module that is registered to process files of that type. The language module in turn is programmed to identify the type and start position of any nested languages allowed in that language. The language module may also identify the end position of the nested language, but may request the assistance of the nested language processor for this task. Once the type and boundaries of a nested language are identified, the compiler framework will pass this portion of the file to the language module registered to process that language type. This process may continue allowing the editor and compiler framework to handle arbitrarily deeply nested languages.

Language Drivers

[0071] If the developer of a language module or custom language editing view wants to expose unique editing features tailored toward a specific language, they can implement a language driver. The language driver encapsulates the unique characteristics of the language and allows them to be plugged directly into the editor without requiring language specific features to be added to the editor itself. The complete API for developing language drivers is described in detail elsewhere herein.

Custom Editors and Views

[0072] Developers that wish to build custom editors for specific languages may do so by creating a class that implements the ISourceDocument interface specified elsewhere herein. The class DefaultSourceDocument can provide a default implementation of all the relevant editor features. Developers may derive their implementation from this class so they only have to override the specific behaviors they want to customize.

[0073] Likewise, developers wishing to build custom views for a specific language may do so by creating a class that implements the ISourceView interface specified also specified elsewhere herein. The class DefaultSourceView can provide a default implementation of all relevant view features. Developers may derive their implementation of ISourceView from this class so they only have to override the specific behaviors they want to customize.

Application Interfaces

Editor Extension API

[0074] A language independent editor can expose a set of APIs that can be used to define custom editor features and custom views for specific languages (e.g., visual editing tools). The full details of this API are described in this section.

Packages	
com.bea.ide.sourceeditor	Defines how the editor can be extended to support custom editing features and views e.g. for a particular language
com.bea.ide.sourceeditor.compiler	Defines ways custom editors or views can interact with and get language information from the compiler
com.bea.ide.sourceeditor.driver	Defines how the editor can be extended with custom language drivers for implementing special language features in the editor
com.bea.ide.sourceeditor.element	Defines how the editor can be extended to

	support new or modified elements in the language. Elements are recognizable pieces of a document (e.g., a function name)
com.bea.ide.sourceeditor.model	Defines ways custom editors or views can interact with the document model. The document model is a data structure that contains the text of a document.
com.bea.ide.sourceeditor.ui	Defines mechanisms developers can use to implement custom UI features

Package com.bea.ide.sourceeditor

This package defines how the editor can be extended to support custom editing features and views e.g. for a particular language. Its classes and interfaces are described below.

public class **com.bea.ide.sourceeditor.SourceUndoManager** extends javax.swing.undo.UndoManager

Constructors

```
public SourceUndoManager(
        ISourceDocument file)
```

Methods

public void discardAllEdits()
Clear all edits in the undo queue. These undos are then lost, and so the edits may not be undone.

public void undo()

public void redo()

```
public void setIgnoreEdits(
        boolean ignoreEdits)
```

```
public boolean addEdit(
        UndoableEdit e)
```

public interface **com.bea.ide.sourceeditor.ISourceView** implements com.bea.ide.document.IDocumentView

The ISourceView interface allows for manipulation of many end-user visible attributes of a document's text view.

Methods

```
public boolean setSelection(
        int lineStart,
        int colStart,
        int lineEnd,
        int colEnd,
        int scrollMode)
```

Selects the indicated text range. To simply move the cursor, rather than select text, pass -1 for lineEnd and colEnd . To goto the start of a line, pass -1 for colStart . For example, selectSource(12, -1, -1, -1) moves the cursor to the start of line 12.

public boolean setSelection(
Range selRange,
int scrollMode)
Selects the range of text indicated.

public boolean setSelection(
ISourceElement el,
int scrollMode)
Selects the specified element. Note that calls to this method, unlike the other setSelection methods, are passed through to @link com.bea.ide.sourceeditor.driver.ISourceViewDriver#getSelectionRange ISourceViewDriver.getSelectionRange to determine the exact text range that should be selected.

public boolean replaceSelectedText(
String strReplacement)
Replaces any selected text with the specified string. If no text is selected, the text will be inserted at the current caret insertion point.

public java.lang.String getSelectedText()
Returns
the text selected in the source view. May return an empty string or null.

public boolean isSingleLineSelection()

public java.lang.String getWordAtCaret()

public com.bea.ide.util.Range getWordRangeAround(
int pos)

public com.bea.ide.util.Range getLineRangeAround(
int pos)

public int getCaretPosition()
Returns
the zero-based character offset of the caret within the view.

public void setCaretPosition(
int pos,
int scrollMode)
Sets the caret position within the source view.
Parameters
pos - the zero-based character offset of the desired caret position
fAutoScroll - specifies whether the source view should scroll, if necessary, to ensure that the caret is visible.

public void setSelectionStart(
int offset,
int scrollMode)
Sets the offset of the start of the source selection. The start and end are always normalized (start <= end).
Parameters
offset - the zero-based character offset of the desired selection start.
fAutoScroll - specifies whether the source view should scroll, if necessary, to ensure that the selection is visible.

public void setSelectionEnd(
int offset,
int scrollMode)

Sets the offset of the end of the source selection. The start and end are always normalized (start <= end).

Parameters

offset - the zero-based character offset of the desired selection end.

fAutoScroll - specifies whether the source view should scroll, if necessary, to ensure that the selection is visible.

public void setSelectionMark(
int offset,
int scrollMode)

Sets the offset of the anchor mark of the source selection.

Parameters

offset - the zero-based character offset of the desired selection mark.

fAutoScroll - specifies whether the source view should scroll, if necessary, to ensure that the selection is visible.

public void setSelectionDot(
int offset,
int scrollMode)

Sets the offset of the movable dot of the source selection.

Parameters

offset - the zero-based character offset of the desired selection dot.

fAutoScroll - specifies whether the source view should scroll, if necessary, to ensure that the selection is visible.

public int getSelectionStart()

Returns

the zero-based offset of the start of the text selection. This is equivalent to the value returned from @link ISourceView#getSelectionEnd <code>getSelectionEnd</code> in the case where there is no selected text.

public int getSelectionEnd()

Returns

the zero-based offset of the end of the text selection. This is equivalent to the value returned from @link ISourceView#getSelectionStart <code>getSelectionStart</code> in the case where there is no selected text.

public javax.swing.text.Caret getCaret()

Returns

the current caret.

public java.lang.String getEditableLineText(
int sourceLine,
Range range)

Returns that portion of a line's text which is not collapsed. This could be null if the entire line is collapsed. It could be just the beginning, end, or middle of the line if it borders one or more collapsed blocks. Or it could be the entire line text if the line is not collapsed.

Parameters

sourceLine - is the 0-based line in the file.

range - is an out-parameter that gets set to the position range that equates to the string. It may be null if it is not requested.

public int getOffset(
Point p)

Returns

offset in the document for the element at the given point, -1 if there is no element at that location

public void showPopupTip(
IPopupTip tip)

Displays a popup tip in an active source editor. If the source view is not active, this method does nothing.

Parameters

tip - the tip to display

offset - the offset associated with the tip. This may or may not be the exact location where the popup appears, depending on several factors including viewport and caret position.

public void addHoverListener(
IHoverListener hoverListener)
Adds a hover listener to this source view. This listener will be notified whenever the mouse hovers over a particular point for a fixed period of time. Note that hover listeners are only useful while the source view is active.

public void removeHoverListener(
IHoverListener hoverListener)
Removes a hover listener from this source view.

Fields

public static final RIGHT_CLICK_POINT_KEY
Key for a cookie on the Application that stores where the user last right clicked in the SourceView

public static final SCROLL_MINIMAL
If a new selection is off the screen, scroll it barely onto the screen.

public static final SCROLL_MIDDLE
If a new selection is off the screen, scroll it to the middle of the screen.

public static final SCROLL_MIDDLE_FORCE
If a new selection is not in the middle of the screen, scroll it to the middle of the screen. That is, selections in the top 2 or bottom 5 visible lines will scroll to the middle.

public interface **com.bea.ide.sourceeditor.ISourceDocument** implements com.bea.ide.document.IDocument

ISourceDocument should be implemented by all document types with text data that wish to perform basic compiler-related functionality such as syntax coloring and parse tree support.

Methods

public IToken getTokenFromOffset(
int offset)
Parameters

offset - a zero-indexed character offset within a document.

Returns

returns the IToken object at the specified offset. This method should never return null, though the token returned may simply indicate that the token information is currently unknown. Note that there is no guarantee that sequential calls to <code>getTokenFromOffset</code> will return the same object or value, as compiler structures are possibly being rebuilt at any time.

public com.bea.ide.element.IElement getElementFromOffset(
int offset)
Parameters

offset - a zero-based character index within the document.

Returns

the parse tree element found at the specified offset. May return null if the document is not parsed at the time that the method is called.

public boolean mergeMetadata(
ISourceElement sourceRootElement)
mergeMetaData is called by the IDE to indicate that a new parse tree is available due to a recompile. Implementors guarantee that after the merge all elements with unchanged

properties remain unchanged, with the exception of range and compiler cookie information, and that appropriate change events are generated for all elements that are not equivalent.

public com.bea.ide.sourceeditor.model.ISourceModel getSourceModel()
Returns
the source model associated with this source document. Once the file is initially loaded, this method should never return null.

Fields

public static final PROP_MID_TWOWAY_EDIT
This property indicates that a document is currently performing a two-way edit. Because two-way edits require multiple interactions between the IDE and compiler threads, there is a short period of time after the edit request during which the document does not fully reflect the change. This property can be used to detect when two-way edits start and complete.

public static final PROP_LAST_COMPILATION_REQUEST
This property is the unique ID (a Long) of the last compilation request sent to the compiler. Listening on this property is useful if an extension requires finer-grained knowledge of compilation state than can be gained by listening to element change events alone. When used with property @link #PROP_LAST_COMPILATION PROP_LAST_COMPILATION, this property can be used to determine when the compiler responds to specific compilation requests.

public static final PROP_LAST_COMPILATION
This property is the unique ID (a Long) of the last compiler request for which the IDE had received a compiler compilation response for this file. Note that this ID may correspond to a compilation request for another file, if this file was recompiled due to a dependency, rather than a direct compilation request. Listening on this property is useful if an extension requires finer-grained knowledge of compilation state than can be gained by listening to element change events alone. When used with property @link #PROP_LAST_COMPILATION_REQUEST PROP_LAST_COMPILATION_REQUEST, this property can be used to determine when the compiler responds to specific compilation requests.

public static final PROP_HAS_ERRORS
This is a boolean property indicating whether the document contains any diagnostics classified as errors.

public static final PROP_HAS_WARNINGS
This is a boolean property indicating whether the document contains any diagnostics classified as warnings.

public class **com.bea.ide.sourceeditor.EditorSvc**

Constructors **public EditorSvc()**

Methods **public static com.bea.ide.sourceeditor.EditorSvc.I get()**

Fields **public static final ATTR_TAB_WIDTH**

public static final ATTR_INDENT_WIDTH

public static final ATTR_FONT_NAME

public static final ATTR_FONT_SIZE

public static final ATTR_FORE_COLOR

public static final ATTR_BACK_COLOR

public static final ATTR_SELECTION_TEXT_COLOR

public static final ATTR_SELECTION_BACK_COLOR

public static final ATTR_BREAKPOINT_BACK_COLOR

public static final ATTR_BREAKPOINT_FORE_COLOR

public static final ATTR_INSTRPOINT_BACK_COLOR

public static final ATTR_INSTRPOINT_FORE_COLOR

public static final ATTR_LINE_NUMBER_COLOR

public static final ATTR_SHOW_LINE_NUMBERS

public static final ATTR_SHOW_MATCHING_CHARS

public static final ATTR_SHOW_WHITESPACE

public static final ATTR_SHOW_SEPARATORS

public static final ATTR_ENABLE_COLLAPSING

public static final ATTR_OVERWRITE
Names the property that stores whether we are in overwrite mode.

public static final ATTR_AUTO_INDENT
Names the property that indicates whether we should auto indent.

public static final ATTR_INSERT_STAR
Names the property that indicates whether we should insert a star after newline within a comment indent.

public static final ATTR_EDIT_READONLY
Names the property that indicates whether we should allow editing of read-only files.

public static final ATTR_SMART_BACKSPACE
Names the property that indicates whether we backspace in indent-sized blocks when the caret is on an indention boundary.

public static final ATTR_INDENT_AFTER_BRACE
Names the property that indicates whether we should add an extra indent for each opening brace on the line before.

public static final ATTR_ALIGN_CLOSING_BRACE
Names the property that determines if we should auto-indent the brace.

public static final ATTR_INSERT_TABS
Names the property that stores whether tabs can be used for indenting.

public static final ATTR_MOUSE_OVER_DELAY
Names the property used for the delay before trying to put up window displaying information about the token (a.k.a. tooltip).

public static final ATTR_PARAM_COMPLETION
Names the property that stores whether parameter completion is enabled.

public static final ATTR_METHOD_COMPLETION
Names the property that stores whether method completion is enabled.

public static final SCROLL_MINIMAL
If a new selection is off the screen, scroll it barely onto the screen.

public static final SCROLL_MIDDLE
If a new selection is off the screen, scroll it to the middle of the screen.

public static final SCROLL_MIDDLE_FORCE
If a new selection is not in the middle of the screen, scroll it to the middle of the screen. That is, selections in the top 2 or bottom 5 visible lines will scroll to the middle.

protected static instance

public static interface **com.bea.ide.sourceeditor.EditorSvc.I**

Methods

public com.bea.ide.sourceeditor.ISourceView getCurrentSourceView()

public com.bea.ide.sourceeditor.ISourceView createSourceView(
ISourceDocument doc)

public com.bea.ide.sourceeditor.model.ISourceModel createSourceModel(
URI uri)

public boolean reloadSourceModel(
ISourceModel model,
URI uri)

public void switchToFile(
URI file,
int lineNumber,
int colNumber,
int colEnd)

public void switchToFile(
URI file,
int lineNumber,
int colNumber,
int colEnd,
int scrollMode)

public class **com.bea.ide.sourceeditor.DefaultSourceViewInfo** extends com.bea.ide.document.DefaultDocumentViewInfo

Utility class allowing for delayed loading of a document's source view. The only reason for this extension of DefaultDocumentViewInfo is to provide a means of checking to see whether the view has been created so that unneccesary view construction can be avoided.

Constructors

public DefaultSourceViewInfo(
ISourceDocument doc,
String strName,
Icon icon,
String strDescription)

Methods **public com.bea.ide.document.IDocumentView createView()**

public boolean isViewCreated()
Returns
whether this document's ISourceView object has been instatiated.

public abstract class **com.bea.ide.sourceeditor.DefaultSourceDocument** extends com.bea.ide.document.DefaultDocument implements com.bea.ide.sourceeditor.ISourceDocument

A default implementation of ISourceDocument . This makes the assumptions that any ISourceElement parse tree is made up of element types deriving from DefaultSourceElement. Usage

Subclasses of this one need only implement @link #createDrivers()??

Constructors **public DefaultSourceDocument(**
Class handler,
URI fileURI)
Creates a new document with empty content using the appropriate editor kit for this file (as provided by the subclass).

Methods **protected javax.swing.undo.UndoManager createUndoManager()**

protected javax.swing.event.UndoableEditListener createUndoableEditListener()

public synchronized void requestTwoWayEdit(
TwoWayEdit edit)
Initiates a two-way edit from a @link com.bea.ide.sourceeditor.compiler.TwoWayEdit TwoWayEdit object. While this method may be called directly, this is almost never done; generally, all requests for two-way edits should go through an @link com.bea.ide.sourceeditor.element.ISourceElement ISourceElement implementation. When called, this method will block the AWT thread until the requested edit completes.
Parameters
edit - The object defining this edit.

public boolean onOpen()

public void onRemove()

protected void uriChanged(
URI uriOld,
URI uriNew)

public java.lang.String getHandlerId()

public com.bea.ide.document.IDocumentViewInfo[] getViewInfo()

public IToken getTokenFromOffset(
int offset)

public boolean isMerging()
Lets external users check to see if the document's parse tree is currently being merged. Tree merging always takes place on the AWT thread, and always blocks that thread from start until completion, so this method is only interesting when called from another thread.

public boolean mergeMetadata(
ISourceElement sourceRootElement)

public com.bea.ide.sourceeditor.model.ISourceModel getSourceModel()

public com.bea.ide.sourceeditor.ISourceView getSourceView()
Gets the ISourceView object associated with this document.

Important: This method will create the source view if it does not exist, and so should be used with extreme caution: creation of views is very expensive, and should only be done if absolutely necessary, generally only when the user specifically requests to see the source view for this document.
Returns
This document's source view.

**protected void setRootElement(
IElement root)**

**public com.bea.ide.element.IElement getElementFromOffset(
int offset)**

public java.io.Reader getReader()

public boolean reload()

protected void createDrivers()
Creates the strategies for this file, calling addStrategy on each.

Fields

public static final PROP_LineNumber

public static final PROP_ColumnNumber

public class **com.bea.ide.sourceeditor.DefaultFileStatus** implements com.bea.ide.ui.statusbar.IFileStatus

A simple object for setting line and column information displayed in the status bar.

Constructors

**public DefaultFileStatus(
IDocument doc)**
Creates the IFileStatus object for the specified document.
Parameters
doc - The file with which this file status object will be associated.

Methods

**public void setLineAndColumn(
int line,
int column)**
Sets the line and column information displayed in the status bar. It is never necessary to call this method directly if the DefaultSourceDocument and built-in ISourceView implementations are used.

public int getLine()
Returns the current line number as displayed in the status bar. It is almost always preferable to call @link com.bea.ide.sourceeditor.ISourceView#getCaretPosition ISourceView.getCaretPosition, and then convert that offset to line and column information via the @link com.bea.ide.sourceeditor.model.ISourceModel ISourceModel interface, since the status bar UI is not necessarily in sync with the actual caret position within the document.
Returns
The current line number as displayed in the status bar.

public int getColumn()
Returns the current column number as displayed in the status bar. It is almost always

preferable to call @link com.bea.ide.sourceeditor.ISourceView#getCaretPosition ISourceView.getCaretPosition, and then convert that offset to line and column information via the @link com.bea.ide.sourceeditor.model.ISourceModel ISourceModel interface, since the status bar UI is not necessarily in sync with the actual caret position within the document.
Returns
The current column number as displayed in the status bar.

Package com.bea.ide.sourceeditor.compiler

This package defines ways custom editors or views can interact with and get language information from the compiler. Its classes and interfaces are described below.

public abstract class **com.bea.ide.sourceeditor.compiler.TwoWayEdit**

A utility class to handle the details of submitting a two-way edit request to the compiler. While this class may be used directly, two-way edits are almost always initiated with methods on the @link com.bea.ide.sourceeditor.element.ISourceElement ISourceElement interface, making direct use of this class unneccesary.

Constructors	**protected TwoWayEdit(ISourceElement elemSource, Object data)**
Methods	**public boolean fire()**
	public abstract boolean doEdit(IStructureFeature feature)
Fields	**protected _elemSource**
	protected _data

public static class **com.bea.ide.sourceeditor.compiler.TwoWayEdit.AddChild** extends com.bea.ide.sourceeditor.compiler.TwoWayEdit

Constructors	**public TwoWayEdit.AddChild(ISourceElement elemParent, ISourceElement elemChild)**
	public TwoWayEdit.AddChild(ISourceElement elemParent, ISourceElement elemChild, int index)
Methods	**public boolean doEdit(IStructureFeature feature)**
Fields	**protected _index**

public static class

com.bea.ide.sourceeditor.compiler.TwoWayEdit.ChangeProperty extends com.bea.ide.sourceeditor.compiler.TwoWayEdit

Constructors	**public TwoWayEdit.ChangeProperty(ISourceElement elemParent, String prop, Object value)**
Methods	**public boolean doEdit(IStructureFeature feature)**

public static class **com.bea.ide.sourceeditor.compiler.TwoWayEdit.RemoveChild** extends com.bea.ide.sourceeditor.compiler.TwoWayEdit

Constructors	**public TwoWayEdit.RemoveChild(ISourceElement elemParent, ISourceElement elemChild)**
Methods	**public boolean doEdit(IStructureFeature feature)**

public class **com.bea.ide.sourceeditor.compiler.JavaBuilder** implements com.bea.ide.sourceeditor.compiler.ILanguageBuilder

The builder class for java document types. This builder will handle construction of IDE-side parse trees for document types made up of java and java annotation languages.

The parse tree constructed is made up of @link com.bea.ide.sourceeditor.element.DefaultSourceElement DefaultSourceElement derivations, and contains most frequently needed parse information. Parse trees constructed by this class do not contain any parse information for java method bodies. This is is done intentionally, for performance reasons.

Construction of the parse tree takes place through calls into the builder methods from the compiler framework. See the @link com.bea.ide.sourceeditor.driver.ICompilerDriver ICompilerDriver for information on how to specify a builder for a different document type.

Constructors	**public JavaBuilder()**
Methods	**public IStructureFeature.IStructureNode createNode(IStructureFeature.StructureCode typeCode, Object structureCookie)**
	public void setProperty(IStructureFeature.IStructureNode node, IStructureFeature.StructureCode typeCode, Object o)
	public void addChild(IStructureFeature.IStructureNode node, IStructureFeature.IStructureNode node1)

public interface **com.bea.ide.sourceeditor.compiler.IPackageCache**

Methods

public java.lang.Object getCacheStatus(
IProject proj)

Returns the status of the package cache. Attempts to access the cache before loading is complete will not cause failures, but they will produce empty or incomplete results.

public java.util.List getMatchingTypes(
IProject proj,
String shortName,
boolean sourceOnly,
boolean prefixMatch,
boolean matchCase)

Returns an list of ITypeDefinition objects corresponding to all types matching the specified type name within the specified IDE project. In the case of type name "List", at least two type definitions will be returned: "java.util.List", and "java.awt.List". Note that only public types will be returned; package private type definitions are ignored.

Parameters

proj - The IDE project in which to look for matching types.

shortName - The short name of the desired type

sourceOnly - Whether non-source classes should be considered. If true, only those classes defined in the specified project's source path will be serached. If false, the project's classpath will be searched as well.

prefixMatch - If true, all types with names beginning with the specified type name will be returned.

matchCase - Indicates whether a case-sensitive check should be used to determine matching types.

public boolean isAmbiguous(
IProject proj,
String typeName,
boolean sourceOnly)

Returns whether there are multiple types that match the non-qualified type name. This method is generally used to determine if a given type should be presented to the user in simple or fully-qualified format.

Parameters

proj - The project in which to look for the type.

typeName - The qualified or non-qualified name of the type in question.

sourceOnly - Whether non-source classes should be considered. If true, only those classes defined in the specified project's source path will be serached. If false, the project's classpath will be searched as well.

Returns

boolean indicating whether the simple name of the specified type is ambiguous within the project.

public java.util.List getMatchingQualifiedTypes(
IProject proj,
String qualifiedTypeName,
boolean sourceOnly)

Returns a list of all types with the specified fully-qualified name. Except in the case of type shadowing, this list should contain zero or one entry(s).

Parameters

proj - The project in which to look for the type.

qualifiedTypeName - The fully-qualified name of the type in question.

sourceOnly - Whether non-source classes should be considered. If true, only those classes defined in the specified project's source path will be serached. If false, the project's

classpath will be searched as well.
Returns
List of all types matching the requested fully-qualified type name.

Fields **public static final CACHE_LOADING**
The cache is actively loading. This process may take some time.

public static final CACHE_LOADED
The cache is loaded and ready to go.

public static interface **com.bea.ide.sourceeditor.compiler.IPackageCache.ITypeDefinition**

ITypeDefinition defines a cached package to be a set of two values, a type name and a package name. For example, type java.util.List is split into type name "List", and package "java.util".

Methods **public java.lang.String getTypeName()**
Returns the type of this package definition. In the case of package java.util.List , this method will return the String "List".

public java.lang.String getPackage()
Returns the package of this package definition. In the case of package java.util.List , this method will return the String "java.util".

public java.net.URI getSourceURI()
Returns the URI of the source file that defines this type. Null will be returned if source is not available.

public java.net.URI getJarURI()
Returns the URI of the classpath jar that contains this type. Null will be returned if the type did not come from a classpath jar.

public interface **com.bea.ide.sourceeditor.compiler.ILanguageBuilder**

this is basically a wrapper for a Javelin IStructureBuilder.

public interface **com.bea.ide.sourceeditor.compiler.ILangObjectMapper**

Interface used to map a class of compiler objects returned from <code>ISourceInfoFeature</code> to objects with the interfaces necessary for IDE related features. Use the extension.xml handler 'urn:com-bea-ide:sourceeditor:sourceinfo' to register an implementation of this interface. Entries have the following format: <object-mapper class="workshop.sourceeditor.lang.TypeInfoObjectMapper"> &lt;compiler-object type="com.bea.compiler.info.IMethod"> <compiler-object type="com.bea.compiler.info.IPackage"> ... &lt;/object-mapper>

Methods **public com.bea.ide.sourceeditor.compiler.IIdentifierInfo mapIdentifierInfo(**
Object info,
ISourceDocument doc,
int offset)
Maps compiler identifier info objects.

public com.bea.ide.sourceeditor.compiler.ICompletionSet mapCompletions(
List completions)
Maps a list of completions to an ICompletionSet implementation. All objects in the list are required to map to the same ILangObjectMapper . Only the first object in the list is used to determin which mapper gets the list.

public interface **com.bea.ide.sourceeditor.compiler.IIdentifierInfo**

Methods

public java.lang.String getShortName()

public java.lang.String getFullName()

public boolean supportsGoToDefinition()

public boolean handleGoToDefinition()

public java.lang.String getDebugExpression()
Gets the expression the debugger should evaluate for its hovering value, null if it isn't a variable

public interface **com.bea.ide.sourceeditor.compiler.ICompletionSet**

Interface representing a set of completions, used for statement completion support.

Methods

public java.util.List getCompletions()
Gets that actuall set of completions. All objects in the list must implement ICompletionItem .

public boolean isPartialCompletion(
String s)
True if the string can represent a partial completion match.

public interface **com.bea.ide.sourceeditor.compiler.ICompletionItem**

Methods

public java.lang.String getDisplayText()
Returns a string containing the text this completion item should show in a drop-down menu. It may be identical to getCompletionText(). But it may optionally include a data type, a parameter list, or anything else. The string may also optionally use tabs for formatting. Using tabs, the drop-down may define any number of virtual columns. The tab stops move as follows: - The first tab is right-aligned at the end of the first column. - The second tab defines a second column, and left-aligns text. - The third tab is right-aligned at the end of the second column. - etc. Text that is left-aligned on one line may overlap right-aligned text on another line. Thus, two completion items with the following display text: public int () String () would lay out as follows in the drop-down menu: +L---------R-L-----------------+ |public int getWidth() | | String doInitialization() | +-----------------------------+ where the L and R marks above show the tab stops.

public com.bea.ide.sourceeditor.compiler.ICompletionItem.IFormatRange[] getDisplayFormatRanges()
Returns an array of IFormatRange objects, which divide the display text up into sub-strings

that should be rendered with alternate fonts or colors, or possibly even with images. Return null to display the entire display text in the default font.

public java.lang.String getCompletionText()
Get the text as we would insert it into the document. This should not include the trigger character (i.e. '.' in Java or '<' in XML). It may however include extra characters after the identifier, that would make the insertion complete (such as "()" after a Java method).

public com.bea.ide.util.Range getCompletionTextInsertPosition()
Assuming the entire completion text has been inserted, including any extra characters tacked on, where should the cursor be placed. Return null for the default behavior setting the cursor at the end. Return a range (may be a single position) if the cursor should be placed before or inside the completion text. For example, between the "()" characters added after a method in Java.

public boolean isChainingCompletion()
Returns true if the source info feature should be queried for more completions immediately following this completion.

public static interface **com.bea.ide.sourceeditor.compiler.ICompletionItem.IFormatRange**

getRange() should return a 0-indexed range of text from getDisplayText that it wants to apply a format to. getTokenName() is used to look up that token name in the syntax coloring feature. As such, the token-names should be the same as what the compiler spits out (i.e. "java.identifier"). getImage() is only used if getTokenName() returns null. getImage() returns an image to display in place of the characters covered by the range.

Methods	**public com.bea.ide.util.Range getRange()**
	public java.lang.String getStyleName()
	public javax.swing.Icon getIcon()
Fields	**public static final BOLD**

public interface **com.bea.ide.sourceeditor.compiler.ICompilerDiagnostic**

Methods

public java.lang.String getCode()
Returns a code for the error.

public java.lang.String getDescription()
Returns the description of the diagnostic.

public int getOffset()
Returns the offset of the diagnostic

public int getLength()
Returns the length of the diagnostic

public java.lang.String getPrescription()
Returns the prescription for the diagnostic or null if none is exists.

	public int getType() Returns the type of this diagnostic.
Fields	**public static final TYPE_ERROR**
	public static final TYPE_WARNING
	public static final TYPE_INFO

public class **com.bea.ide.sourceeditor.compiler.CompilerSvc**

The Compiler Service is the general mechanism for communication between the IDE and Javelin, BEA's compiler framework. Since this type of communication is extremely structured, occuring primarily when the file system or document contents change, it is rarely necessary to call many of these methods explicity in an IDE extension. Some, like @link com.bea.ide.sourceeditor.compiler.CompilerSvc.I#getFeature getFeature are frequently called by external consumers.

Constructors	**public CompilerSvc()**
Methods	**public static com.bea.ide.sourceeditor.compiler.CompilerSvc.I get()**
Fields	**public static final SYNCHRONOUS_TWOWAY_EDITS**
	protected static instance

public static interface **com.bea.ide.sourceeditor.compiler.CompilerSvc.ITwoWayEditLock**

Methods	**public boolean runMerger()** call 'runMerger' to merge the new parse trees and update the IDE's error list. This call should ONLY be made on the AWT thread. **Returns** boolean indicating success or failure. This method should only fail in the case of catastrophic application or compiler failure.

public static interface **com.bea.ide.sourceeditor.compiler.CompilerSvc.ICompilerLockManager**

Methods	**public void acquireReadLock()** Acquires a read lock for the compiler project. Note that this lock MUST be released when the client is done with the necessary compiler operations.
	public void releaseReadLock() Releases the comiler read lock.

public static interface **com.bea.ide.sourceeditor.compiler.CompilerSvc.I**

Methods

public java.lang.Object getProperty(
  String name,
  Object defValue)

Parameters
  name - a property key

Returns
  a global compiler property

public java.lang.Object getProperty(
  IProject proj,
  String name,
  Object defValue)

Parameters
  proj - an IDE project in the current workspace
  name - a property key

Returns
  a project specific compiler property

public boolean hasCompiler(
  IProject proj)

Determines whether a specific IDE project has an active compiler project.

Parameters
  proj - and IDE project in the current workspace

Returns
  true if there is an associated compiler project

public com.bea.compiler.IProject getProject(
  IProject proj)

todo: remove this. we shouldn't be exposing the compiler IProject.

Parameters
  proj - an IDE project in the current workspace

Returns
  the compiler project associated with a specific IDE project

public java.util.Iterator getLanguages()

Returns
  an iterator of com.bea.compiler.ILanguage objects currently loaded by the compiler.

public boolean hasLanguage(
  String ext)

Returns true if the given file extension has a default language registered with the compiler

Returns
  true if the compiler has a language registered for the given extension

public boolean isFilenameRestricted(
  String ext)

Returns true if files with this extension have a restricted namespace in the filesystem. It is assumed that this namespace is shared with all other extensions that are also restricted. This check is done by seeing if the language registered for this extension is a name restricted language.

public boolean sameLanguage(
  String ext1,
  String ext2)

Determines whether or not two extensions have the same default language

Returns
  true if the extensions have the same default language

public boolean registerExtension(
String extNew,
String extExisting)
Registers the given new extension to have the same language as the existing extension. Note that if the existing extension does not have a language associated with it or if the new extension already has an existing language, this method will do nothing and return false.
Returns
true if the new extension now has the same language as the existing extension, false otherwise.

public boolean isFeatureAvailable(
ISourceDocument doc,
Class featureClass)
Determines if a compiler feature is available for a given document.
Parameters
doc - the document associated with the desired feature instance
featureClass - the class of the desired feature. For example, <code>IStructureFeature.class</code>
Returns
true if the feature is valid for the document.
See Also
com.bea.ide.sourceeditor.driver.ICompilerDriver#getAvailableFeatures

public java.lang.Object getFeature(
ISourceDocument doc,
Class featureClass)
Allows access to compiler features associated with a specific document. Note that the only allowable feature classes for a paticular document type are those explicitly returned from the document's ICompilerDriver's @link com.bea.ide.sourceeditor.driver.ICompilerDriver#getAvailableFeatures getAvailableFeatures call.
Parameters
doc - the document associated with the desired feature instance
featureClass - the class of the desired feature. For example, <code>IStructureFeature.class</code>
Returns
the requested IFeature, if valid and available. May return null.

public java.lang.Object getFeature(
IProject proj,
Class featureClass)
Allows access to compiler features associated with a specific IDE project.
Parameters
proj - the IDE project
featureClass - the class of the desired feature. For example, <code>ITokenInfoFeature.class</code>
Returns
the requested IFeature, if valid and available. May return null.

public void registerDocument(
ISourceDocument doc)
Associates the given source document with it's URI. The compiler service will rely on the source document to notify it of important changes and will defer to the file system listener on the file to keep it informed about things like whether or not the file has been deleted.

public void unregisterDocument(
URI uri,
ISourceDocument doc)
Remove the association between the given URI and the given document. If no such association exists, this method does nothing. Document object should call this method in their finalizers to insure that any compiler data structures associated with the document are removed.

public void setFileModified(
ISourceDocument doc,
ISourceModelEvent e,
ISourceModelEvent.EventType type)

Called by various document change listeners to notify the compiler of changes to document content. Generally, only actual content changes (those that would require a recompile) need to be passed to the compiler. Under normal circumstances, it should not be necessary to call this method explicitly.

Parameters

doc - the document that has been modified
e - the event associated with the modification
type - the type of modification

public void refreshFile(
ISourceDocument doc)

Initiates a full recompile of the specified document. This will cause a refresh of all compiler features, including the IStructureFeature (i.e., the IDE parse tree).

Parameters

doc - the file to be recompiled

public void refreshFile(
URI path)

Initiates a full recompile of the specified URI. This will cause a refresh of all compiler features, including the IStructureFeature (i.e., the IDE parse tree).

Parameters

path - the file to be recompiled

public com.bea.ide.sourceeditor.compiler.CompilerSvc.ITwoWayEditLock getEditLockObject()

Used by the two-way edit mechanism to ensure that the AWT thread waits for two-way edits to complete before continuing application execution.

public com.bea.ide.sourceeditor.compiler.CompilerSvc.ICompilerLockManager getLockManager(
URI uri)

Retrieves a lock manager for the compiler project associated with the specified URI. Compiler locks are required when accessing compiler features at a time when compilation may take place.

Parameters

uri - The uri used to determine which project should be locked.

public com.bea.ide.sourceeditor.compiler.CompilerSvc.ICompilerLockManager getLockManager(
IProject proj)

Retrieves a lock manager for the compiler project associated with the specified project. Compiler locks are required when accessing compiler features at a time when compilation may take place.

Parameters

proj - The IDE project used to determine which compiler project should be locked.

public boolean canCompileFile(
URI uri)

Checks to see if the URI points to a valid, compilable file.

Parameters

uri - the URI pointing to the file in question.

public java.net.URI getPathForType(
ISourceDocument doc,
String type)

Get the path to the definition of a type in the context of the given document. Note: this will block until the compiler finishes it's current work.

Parameters

doc - The source document in whose context the type will be resolved.

type - The fully-qualified type whose methods you want to find.

Returns

The URI to the definition of the type.

public java.util.List getMethodsForType(
ISourceDocument doc,
String type,
boolean includeSupers)

Get the methods on a given class/interface. Note: this will block until the compiler finishes it's current work.

Parameters

doc - The source document in whose context the type will be resolved.

type - The fully-qualified type whose methods you want to find.

includeSupers - If false, only methods directly on the given type are returned, otherwise all methods available on that type and it's supertypes are given.

Returns

A List of JavaMethod objects corresponding to the methods. These are somewhat raw JavaMethods, and do not support editing. If you need full information, you'll need to open the appropriate document and get it's parse tree. This list may be empty, but is never null.

public java.util.List getFieldsForType(
ISourceDocument doc,
String type,
boolean includeSupers)

Get the fields on a given class/interface. Note: this will block until the compiler finishes it's current work.

Parameters

doc - The source document in whose context the type will be resolved.

type - The fully-qualified type whose fields you want to find.

includeSupers - If false, only methods directly on the given type are returned, otherwise all methods available on that type and it's supertypes are given.

Returns

A List of JavaVariables objects corresponding to the fields. These are somewhat raw JavaFields, and do not support editing. If you need full information, you'll need to open the appropriate document and get it's parse tree. This list may be empty, but is never null.

public java.util.List getSupersForType(
ISourceDocument doc,
String type)

Get the superclass and superinterfaces of a given class/interface. Note: this will block until the compiler finishes it's current work.

Parameters

doc - The source document in whose context the type will be resolved.

type - The fully-qualified type whose supers you want to find.

Returns

A List of Strings corresponding to the superclass (unless it's java.lang.Object) and all declared interfaces.

public com.bea.ide.sourceeditor.compiler.IIdentifierInfo getIdentifierInfo(
ISourceDocument doc,
int offset)

Get identifier info for an offset in a source document.

Parameters

doc - The source document for which to get the identifier info.

offset - The offset at which the identifier is located.

public com.bea.ide.sourceeditor.compiler.IPackageCache getPackageCache()

Gets the package cache: this object is used to get general package information, such as that used for generating auto-import suggestions and generating searchable package/class name lists.

public com.bea.ide.sourceeditor.driver.ILanguageDriver getDefaultLanguageDriver(
  ISourceDocument doc,
  String languageName)

Returns a default language driver implementation appropriate to the language at the specified offset within the specified document. This method should only be called by @link com.bea.ide.sourceeditor.driver.ISourceViewDriver#getLanguageDriver ISourceViewDriver.getLanguageDriver implementations that do not want to customize language behavior. All other callers should obtain an ILanguageDriver by calling through their document's @link com.bea.ide.sourceeditor.driver.ISourceViewDriver#getLanguageDriver ISourceViewDriver.getLanguageDriver method. Note that a valid return from this method does not guarantee that the specified langauge exists within the document: @link #getLanguageName getLanguageName and @link #getLanguageStack getLanguageStack should be used to determine which languages are present.

Parameters

  doc - The document upon which to get the language driver
  languageName - The language for which a driver is needed.

Returns

  an ILanguageDriver implementation. Null if the language does not exist.

public java.lang.String getLanguageName(
  ISourceDocument doc,
  int offset)

Returns the name of the active language at the specified offset in the specified document.

Parameters

  doc - The document. See also @link #getLanguageStack getLanguageStack.
  offset - The position within the document.

Returns

  The name of the language at the specified location. Null is the location is invalid.

public java.lang.String[] getLanguageStack(
  ISourceDocument doc,
  int offset)

Returns an array of language names corresponding to the stack of languages present in the specified document at the specified offset. See also @link #getLanguageName getLanguageName.

Parameters

  doc - The document
  offset - The position within the document

Returns

  An array of strings corresponding to the language stack. Index zero corresponds to the top-level language of the document, index length - 1 corresponds to the langauge active at the offset. Null is the offset is invlaid.

public void loadCompleteSource(
  IProject proj)

Loads all source files in the specified project as quickly as possible. Under normal circumstances, source files are loaded incrementally, so as to provide maximum responsiveness to the user. However, certain features, such as auto-correction, are not available until all source files have been loaded. Calling this method will load all source files, making these features available sooner, possibly at the expense of foreground task responsiveness.

Parameters

  proj - The project for which to load all source files.

public com.bea.ide.filesystem.IFileFilter getCompileFilter()

get a file filter that only accepts compilable files

Returns

  IFileFilter implementation that filters out files that cannot be compiled by the compiler svc.

public com.bea.ide.ui.propertyeditor.IPropertyViewDriver getPropertyViewDriver(

IDocument document)
Gets an IPropertyViewDriver for the given document.
Parameters
document - the document for which the driver should be created.
Returns
the driver to get properties for the given document, or null if no driver is available for this document.

public boolean outputDiagnostics(
com.bea.compiler.IProject project,
List uris,
OutputSvc.IOutputWindow window)
Outputs the compiler diagnostics for the file into the specified OutputWindow.
Parameters
project - project that owns the files
uris - List of file URIs to show diagnostics for
window - IOutputWindow in which to display diagnostics
Returns
If messages were generated, this method returns true. False means no errors were present in the files.

public java.util.List getDiagnostics(
ISourceDocument doc,
Range range)
Returns a list of the @link ICompilerDiagnostic ICompilerDiagnostic objects found in the specified document in the specified range. Note that diagnostics overlapping the range will be returned as well as those that are completely contained.
Parameters
doc - the source document for which diagnostics are desired
range - the range in the document text that should be searched
Returns
A list of @link ICompilerDiagnostic ICompilerDiagnostic objects.

public boolean addFileStore(
Object o)
Add a com.bea.compiler.IFileStore to the running Javelin compiler instance.
Parameters
o - the IFileStore object to add
Returns
Success or failure. This method will fail if the object was not of the correct type.

public com.bea.ide.sourceeditor.compiler.ICompletionSet mapCompletions(
List completions)
Maps a list of completions of various types to an ICompletionSet that the source editor understands. This is necessary for completion types like IAttributeValue where on completion set will wrap another.
Parameters
completions - a list of compiler supplied competion objects.
Returns
a completion set.

Package com.bea.ide.sourceeditor.driver

This package defines how the editor can be extended with custom language drivers for implementing special language features in the editor. Its classes and interfaces are described below.

public class **com.bea.ide.sourceeditor.driver.SourceInstructionPoint** implements com.bea.ide.debug.IDebugInstructionPoint

Implements the IDebugBreakPoint interface for text files.

Constructors

public SourceInstructionPoint(
IDocument file,
int line,
int type)

Methods

public com.bea.ide.document.IDocument getFile()

public int getLine()

public int getType()

public class **com.bea.ide.sourceeditor.driver.SourceBreakpoint** extends com.bea.ide.debug.DefaultBreakpoint implements com.bea.ide.debug.ISourceBreakPoint

This class implements a breakpoint that will work on a file that implements ISourceDocument.

Constructors

protected SourceBreakpoint(
IDocument file)

public SourceBreakpoint(
IDocument file,
int line)

Methods

public void documentLoaded()
In init, this breakpoint is about to be associated with an actual file. So we should open the file, and grab an element.

public void documentUnloaded()
In the case that a file is going away, we need to make sure to persist the current offset info.

public void updateLine()

public int getLine()

public void reset(
Element elt)

public void syncBreakpoint(
DebugCommands dbgCmds)

public com.bea.ide.document.IDocument getFile()

public java.net.URI getURI()

public java.lang.String getExtension()

public com.bea.ide.debug.IDebugInstructionPoint breakpointHit()

public java.lang.String toString()

Fields

protected _file

protected _lineNum

protected transient _line

protected transient _model

public static class **com.bea.ide.sourceeditor.driver.SourceBreakpoint.DefaultLine** implements com.bea.ide.sourceeditor.model.ISourceModel.Line

Constructors	**public SourceBreakpoint.DefaultLine(Element el)**
Methods	**public int getStartOffset()** **public int getEndOffset()** **public java.lang.String getText()**

public interface **com.bea.ide.sourceeditor.driver.ISourceViewDriver** implements com.bea.ide.core.IDriverSupport

Implements customizable features of the source view. This driver extends IDriverSupport so that the source view can obtain customized drivers for structure view, property view, etc.... The SourceView will call ISourceViewDriver#getDriver(Object) for drivers not implemented by the source view itself. Drivers that the source view implements, include: IHelpDriver, IPrintDriver and ITransferDriver.

Methods

**public com.bea.ide.util.Range getSelectionRange(
ISourceElement element)**
Allows a document type to specify what range of text should be selected when an element selection is translated to a text selection, generally during a call to @link com.bea.ide.sourceeditor.ISourceView#setSelection(ISourceElement) ISourceView.setSelection . For example, the text selection that results from double-clicking an element in the structure pane is determined by calling this method. The default selection places the cursor at the beginning of the element. Other implementations may want to select the entire element or a sub-element or property.

public boolean showCollapse()
Specifies whether collapsible editor sections are displayed in the source editor line bar. If a particular document type does not specify any collapsible element types in @link ISourceViewDriver#isCollapsible isCollapsible then this method should return false, as extra processing is required when the feature is enabled.
Returns
a boolean indicating whether collapsible sections are enabled for the owning document type.

**public boolean isCollapsible(
ISourceElement element)**
Allows a document implementation to specify which ISourceElement types are collapsible. Note that this method is only relevant if @link ISourceViewDriver#showCollapse showCollapse returns true.
Parameters
element - The element in question. Generally only the element's type is considered.

public boolean isCollapsedByDefault(
ISourceElement element)

Allows a document implementation to specify which ISourceElements are collapsed by default. Note that this method is only relevant if @link ISourceViewDriver#showCollapse showCollapse returns true.

Parameters

element - The element in question. Generally only the element's type is considered.

public java.lang.String getCollapsedText(
ISourceElement element)

Allows a document implementation to specify the text that appears on the single line left visible when an editor section is collapsed. Note that this method is only relevant if @link ISourceViewDriver#showCollapse showCollapse returns true.

Parameters

element - The element in question.

public com.bea.ide.sourceeditor.element.ISourceElementGroup getElementGroup(
ISourceElement element)

Some elements are not themselves collapsible, but can be grouped with like elements into an imaginary container element that is collapsible.

Parameters

element - The element in question.

public boolean isCollapsible(
ISourceElementGroup group)

Allows a document implementation to specify which ISourceElement types are collapsible as groups. Note that this method is only relevant if @link ISourceViewDriver#showCollapse showCollapse returns true.

Parameters

group - The group type in question.

public boolean isCollapsedByDefault(
ISourceElementGroup group)

Allows a document implementation to specify which ISourceElement groups are collapsed by default. Note that this method is only relevant if @link ISourceViewDriver#showCollapse showCollapse returns true.

Parameters

group - The element group type in question.

public java.lang.String getCollapsedText(
ISourceElementGroup group)

Allows a document implementation to specify the text that appears on the single line left visible when an editor section is collapsed. Note that this method is only relevant if @link ISourceViewDriver#showCollapse showCollapse returns true.

Parameters

group - The element group in question.

public com.bea.ide.sourceeditor.driver.ILanguageDriver getLanguageDriver(
String language)

Returns an ILanguageDriver appropriate for the named language. Language names are generally obtained by calling @link com.bea.ide.sourceeditor.compiler.CompilerSvc.I#getLanguageName CompilerSvc.get().getLanguageName.

Parameters

language - a the name of the language for which a driver is needed

Returns

an ILangaugeDriver specific to the specified language

public interface **com.bea.ide.sourceeditor.driver.INavigationBarDriver**

Contains file-specific code for implementing the navigation bar. Note that each file could potentially show up in the navigation bar either as the primary file or as a control being used.

Methods

public boolean isType(
IElement node)
Determines whether the given node represents a type to be shown in the types list.

public boolean isMember(
IElement node)
Determines whether the given node represents a member of the enclosing type to be shown in the members list.

public boolean isControl(
IElement node)
Determines whether the given node represents a control that is being used by this type. (Note that controls should be symblic links.)

public boolean isEvent(
IElement node)
Determines whether the given node represents an event that is exported by this type (for use as a control).

public boolean isEventHandler(
IElement node)
Determines whether the given node represents an event handler for an event thrown by a control used by this type.

public javax.swing.Icon getIcon(
IElement b)
Returns the icon to be used for the given node. It may be showing up in either the types or members list, for the current file or as a control being used.

public com.bea.ide.element.IElement getControlForHandler(
IElement handler)
Returns the node for the control corresponding to the given handler.

public com.bea.ide.element.IElement getEventForHandler(
IElement handler)
Returns the node for the event that is caught by the given handler. This may return null if it is a handler for an event that is not thrown.

public com.bea.ide.element.IElement getHandlerForEvent(
IElement type,
IElement control,
IElement event)
Returns the handler in the given type for the given event. This should create the handler if it does not exist.

public interface **com.bea.ide.sourceeditor.driver.ILanguageDriver**

Implementations of this driver provide customization for working with the languages in the source view. The methods require understanding of the lexical structure of the language for correct implemention. Note that one document may make use of several language drivers, since a single document type may contain several nested languages.

Methods

public void setDocument(
IDocument doc)

Sets the source document upon which this document will act. This method will be called immediately after instantiation.

public boolean isOpenChar(
    char ch)
Determines whether the given character might open a block.

public boolean isCloseChar(
    char ch)
Determines whether the given character might close a block.

public int getOpenBlockCount(
    ISourceModel.Line line,
    ILanguageDriver.IBlockDelim delim,
    int length)
Returns the number of non-matching open-blocks on the given line. The block end given will be the open-block of the pair. The length parameter is the number of character in the line's text that should be examined for open blocks.
Parameters
    line - Line to examine for open blocks.
    delim - Type of block to scan for.
    length - Length of <code>line</code> to examine from start.
Returns
    Number of non-matching open block delimiters on first <code>length</code> characters of <code>line</code>.

public com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockInfo findMatchingOpenBlock(
    int pos,
    char ch)
Returns the block whose open delimiter matches that found at the position given. If neither delimiter is found null is returned. If a mismatch is found then a block is found with only the close delimiter. Use the faster short circuit isCloseChar method first.
Parameters
    pos - Offset of character <code>ch</code> in document.
    ch - Close character.
Returns
    Corresponding block object, null if not found.

public com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockInfo findMatchingCloseBlock(
    int pos,
    char ch)
Returns the block whose close delimiter matches that found at the position given. If neither delimiter is found null is returned. If a mismatch is found then a block is found with only the close delimiter. Use the faster short circuit isOpenChar method first.
Parameters
    pos - Offset of character <code>ch</code> in document.
    ch - Close character.
Returns
    Corresponding block object, null if not found.

public boolean parameterHelpAvailable(
    int offset,
    char typed)
Called after a user types a character in the source editor to determine if the specified character may initiate parameter popup help.
Parameters
    offset - the offset of the caret
    typed - the potential parameter help initiation character

public com.bea.ide.sourceeditor.driver.ILanguageDriver.IParameterHelp[]

getParameterHelp(
  int offset,
  char typed)
Returns an object used to define the UI that will be used for the parameter help popup.

public boolean suppportsCtrlHover(
  String tokenTypeName)
Allows this driver to define which types of tokens support ctrl-hover and ctrl-click definition information features. If this method returns true for a given token type, more expensive calls utilizing the object mappers defined in the IDE extension's "urn:com-bea-ide:sourceeditor:sourceinfo" XML extension will be used to determine if definition information is actually available.
Parameters
  tokenTypeName - the language specific name for a particular token type, such as "java.identifier".
Returns
  boolean indicating whether tokens of this type ever support ctrl-hover behavior.

public boolean canAutoCorrect(
  String diagnosticCode)
Allows this driver to specify the types of errors for which it can offer auto-corrections. Calls to this method should be very fast, as it may be called many times during each compilation notification. If an error is correctable, this driver may or may not ever receive a request for the actual IAutoCorrection object via @link #getAutoCorrection getAutoCorrection.
Parameters
  diagnosticCode - The type of error found.
Returns
  a boolean indicating whether this driver can correct the specified error.

public com.bea.ide.sourceeditor.driver.ILanguageDriver.IAutoCorrection getAutoCorrection(
  String diagnosticCode,
  int offset,
  int length,
  ILanguageDriver.IAutoCorrection innerCorrection)
Allows this driver to returns auto-correction suggestions for various errors found in the source code. If this driver is not able to provide a correction, the inner language's correction (if present) should be returned, unless this driver wants to explicitly hide that correction.
Parameters
  diagnosticCode - The type of error found.
  offset - The location of the error within this driver's document.
  length - The length of text to which the error applies.
  innerCorrection - The auto-correction provided by inner language nodes, if any. This parameter is frequently null.
Returns
  An IAutoCorrection object corresponding to corrections for the specified error.

public java.net.URI[] getHelpContextList(
  Range range)
Returns the list of context help URIs for the given range.

public com.bea.ide.sourceeditor.driver.ILanguageDriver.IAutoFill getAutoFill(
  int offset,
  char typed)
Allows the driver to return an auto-fill text range replacement for keystrokes sent to the source view. If this driver doesn't wish to provide auto-fill it can return <code>null</code> in which case a default action will be taken for the character. A typical auto-fill example is for indenting new lines, e.g. " -> "indent". <p>This method is only called for printable characters, e.g. > 0x20, and for the new-line character ". CONSIDER:This doesn't allow access to modifier keystrokes, CTRL, ALT, etc. Should we have another method, say public InputMap getKeymap()?
Parameters

offset - The location of the caret within this driver's document.
typed - The character typed.

Returns

An auto-fill replacement object, null for default action.

public static interface **com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockInfo**

Summary of the information gathered by the findMatchingXxx methods.

Methods

public int getType()

public com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockDelim getOpenDelim()

public com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockDelim getCloseDelim()

public boolean isMismatched()

Fields

public static final TYPE_MISMATCHED
Indicates an undefined block.

public static final TYPE_SIMPLE
Simple block type, e.g. () or [] in Java/C++/C.

public static final TYPE_STRUCTURE
Structural block type, e.g. in Java/C++, in HTML.

public static interface **com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockDelim**

Defines one end of a block. Note that block close positions are inclusive unlike ranges which have an exclusive end position.

Methods

public int getOpenPos()

public int getClosePos()

public java.lang.String getToken()

public static interface **com.bea.ide.sourceeditor.driver.ILanguageDriver.IParameterHelp**

Methods

public java.lang.String getDisplayPrefix()

public java.lang.String getDisplaySuffix()

public java.lang.String[] getParameterDescriptions()

public java.lang.String getParameterSeparator()

public boolean isPrimary()

public static interface **com.bea.ide.sourceeditor.driver.ILanguageDriver.IAutoCorrection**

Methods

public java.lang.String[] getTipText()
Returns the text to show in the correction pup-up.

public boolean doCorrection(
    int item)
Execute the correction chosen by the user.
Parameters
    item - an index into the original array of choices returned from getTipText().

public static interface **com.bea.ide.sourceeditor.driver.ILanguageDriver.IAutoFill**

Object describing an auto-fill. Each auto-fill has three components, the replacement text, a replacement range and the final caret position.

Methods

public java.lang.String getText()
Returns the auto-fill text.
Returns
    Auto-fill text.

public com.bea.ide.util.Range getRange()
Auto-fill replacement range.
Returns
    Range to be replaced with auto-fill text.

public int getCaretPosition()
Position the caret should be left at after the auto-fill range has been replaced.
Returns
    Caret position.

public interface **com.bea.ide.sourceeditor.driver.ICompilerDriver**

ICompilerDriver implementations allow different document types to customize their behavior with respect to the Javelin compiler framework. See individual methods for more information.

Methods

public com.bea.ide.sourceeditor.compiler.ILanguageBuilder createBuilder()
Allows an IDocument implementation to return an ILanguageBuilder that will be used for construction of its parse tree. For example, @link com.bea.ide.lang.java.driver.JavaCompilerDriver#createBuilder JavaCompilerDriver returns a @link com.bea.ide.sourceeditor.compiler.JavaBuilder JavaBuilder instance, which is used to build the standard java parse tree.
Returns
    an object implementing the ILangaugeBuilder interface.

public java.lang.Class[] getAvailableFeatures()
Allows an IDocument implementation to specify which compiler features should be supported for that document type. Features specified by this method may be accessed via @link com.bea.ide.sourceeditor.compiler.CompilerSvc.I#getFeature CompilerSvc.get().getFeature.
Returns
an array of classes that implement the @link com.bea.compiler.feature.IFeature IFeature interface.

public boolean isFeatureAvailable(
Class featureClass)
Determines whether the given feature is in the available features list for this driver.
Parameters
featureClass - The class object for the feature in question.
Returns
true if featureClass is an available feature.

public class **com.bea.ide.sourceeditor.driver.DefaultSourceViewDriver** extends com.bea.ide.sourceeditor.driver.DefaultDocumentDriver implements com.bea.ide.sourceeditor.driver.ISourceViewDriver

A default implementation of SourceViewStrategy .

Constructors

public DefaultSourceViewDriver(
ISourceDocument moduleDoc)

Methods

protected void addDriver(
Class oClass,
Object driver)
Allows derived classes to add a driver to the driver map. These will be returned automatically in response to a call to getDriver()

public java.lang.Object getDriver(
Class oClass)

public com.bea.ide.util.Range getSelectionRange(
ISourceElement element)

public boolean showCollapse()

public boolean isCollapsible(
ISourceElement element)

public boolean isCollapsedByDefault(
ISourceElement element)

public java.lang.String getCollapsedText(
ISourceElement element)

public com.bea.ide.sourceeditor.driver.ILanguageDriver getLanguageDriver(
String language)

public com.bea.ide.sourceeditor.element.ISourceElementGroup getElementGroup(
ISourceElement element)

public boolean isCollapsible(
ISourceElementGroup group)

public boolean isCollapsedByDefault(
ISourceElementGroup group)

public java.lang.String getCollapsedText(
ISourceElementGroup group)

public class **com.bea.ide.sourceeditor.driver.DefaultLanguageDriver** extends com.bea.ide.sourceeditor.driver.DefaultDocumentDriver implements com.bea.ide.sourceeditor.driver.ILanguageDriver

Constructors	**public DefaultLanguageDriver()**
Methods	**public com.bea.ide.sourceeditor.driver.ILanguageDriver.IParameterHelp[] getParameterHelp(int offset, char typed)**
	public boolean parameterHelpAvailable(int offset, char typed)
	public boolean isOpenChar(char ch)
	public boolean isCloseChar(char ch)
	public int getOpenBlockCount(ISourceModel.Line line, ILanguageDriver.IBlockDelim delim, int length)
	public com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockInfo findMatchingOpenBlock(int pos, char ch)
	public com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockInfo findMatchingCloseBlock(int pos, char ch)
	public boolean suppportsCtrlHover(String tokenTypeName)
	public boolean canAutoCorrect(String diagnosticCode)
	public com.bea.ide.sourceeditor.driver.ILanguageDriver.IAutoCorrection getAutoCorrection(String diagnosticCode, int offset, int length, ILanguageDriver.IAutoCorrection innerCorrection)
	public java.net.URI[] getHelpContextList(Range range)

public com.bea.ide.sourceeditor.driver.ILanguageDriver.IAutoFill getAutoFill(
int offset,
char typed)

public static class **com.bea.ide.sourceeditor.driver.DefaultLanguageDriver.BlockInfo** implements com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockInfo

Constructors	**public DefaultLanguageDriver.BlockInfo()**
Methods	**public int getType()**
	public com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockDelim getOpenDelim()
	public com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockDelim getCloseDelim()
	public void setType(int type)
	public void setOpenDelim(ILanguageDriver.IBlockDelim openDelim)
	public void setCloseDelim(ILanguageDriver.IBlockDelim closeDelim)
	public boolean isMismatched()

public static class **com.bea.ide.sourceeditor.driver.DefaultLanguageDriver.BlockDelim** implements com.bea.ide.sourceeditor.driver.ILanguageDriver.IBlockDelim

Constructors	**public DefaultLanguageDriver.BlockDelim()**
Methods	**public int getOpenPos()**
	public int getClosePos()
	public java.lang.String getToken()
	public void setOpenPos(int pos)
	public void setClosePos(int pos)
	public void setToken(String token)
	public boolean equals(DefaultLanguageDriver.BlockDelim delim)

public class **com.bea.ide.sourceeditor.driver.DefaultDocumentDriver**

Constructors

public DefaultDocumentDriver(
IDocument sourceDoc)

Methods

public void setDocument(
IDocument sourceDoc)

public com.bea.ide.document.IDocument getDocument()

public class **com.bea.ide.sourceeditor.driver.DefaultDebugRunnable** extends com.bea.ide.sourceeditor.driver.DefaultDocumentDriver implements com.bea.ide.workspace.IWebRunnable

This is the debug class to take care of being runnable, etc. UNDONES(willpugh) All the funky JSX/Script environments stuff should be pulled out into a script DebugRunnable class for jsx.

Constructors

public DefaultDebugRunnable(
ISourceDocument file,
boolean runnable,
boolean buildable)

Methods

public boolean isRunnable()

public java.net.URL getRunnableUrl()

public boolean isBuildable()

public class **com.bea.ide.sourceeditor.driver.DefaultDebugDriver** extends com.bea.ide.sourceeditor.driver.DefaultDocumentDriver implements com.bea.ide.debug.IDebugDriver

Implements the strategy for debugging text files.

Constructors

public DefaultDebugDriver(
ISourceDocument file)

Methods

public com.bea.ide.debug.ISourceBreakPoint createBreakPoint(
IDocument file,
int line)
This creates a breakpoint with the given URI + line number. This will not actually be associated with a line in a document, until addBreakPoint is called.

public com.bea.ide.debug.ISourceBreakPoint createMethodBreakPoint(
IDocument file,
String className,
String methodName,
String[] params)

public boolean addBreakPoint(
  ISourceBreakPoint bp)

public void removeBreakPoint(
  ISourceBreakPoint bp)

public boolean hasBreakPoint(
  IDocument idfile,
  int line)

public com.bea.ide.debug.IDebugInstructionPoint setInstructionPoint(
  IDocument idfile,
  int line,
  int type)

public void unsetInstructionPoint(
  IDebugInstructionPoint ip)

public boolean hasInstructionPoint(
  IDocument idfile,
  int line)

public class **com.bea.ide.sourceeditor.driver.DefaultCompilerDriver** implements com.bea.ide.sourceeditor.driver.ICompilerDriver

The default compiler driver specifies default compiler interactions for compiler-recognized document types. This behavior generally consists of syntax coloring and error reporting within the source editor. This default behavior does NOT include the creation of an IDE-side parse/structure tree. For this functionality, a document type must supply a compiler driver implementation that returns a valid @link com.bea.ide.sourceeditor.compiler.ILanguageBuilder ILanguageBuilder interface from the @link com.bea.ide.sourceeditor.driver.ICompilerDriver#createBuilder createBuilder method. See @link com.bea.ide.lang.java.driver.JavaCompilerDriver JavaCompilerDriver for an example.

Note that a providing a compiler driver on a document type is only useful if that document type is recognized by the compiler framework. This functionality is generally achieved by writing a compiler language extension or by mapping a new document type to an existing language extension via the compiler's 'javelin-config.xml' configuation file.

Constructors

public DefaultCompilerDriver(
  ISourceDocument doc)

Methods

public com.bea.ide.sourceeditor.compiler.ILanguageBuilder createBuilder()
Returns a builder object used by the compiler to generate IDE-side parse trees. By default, document types do not supply a builder.
Returns
  null

public java.lang.Class[] getAvailableFeatures()
Returns the set of compiler features that should be made available on this document. The default set includes the following:

Token Info Feature : allows access to tokenization information for this driver's document.

Node Info Feature : allows access to langauge transition information for this driver's document.

Source Info Feature : allows access to source editor features such as auto-completion.

Structure Info Feature : allows access to structure tree information for this driver's document. Note that while this feature is supported by default, it is only useful to a @link DefaultCompilerDriver DefaultCompilerDriver derivation that returns a valid @link com.bea.ide.sourceeditor.compiler.ILanguageBuilder ILanguageBuilder interface from the @link com.bea.ide.sourceeditor.driver.ICompilerDriver#createBuilder createBuilder method. See @link com.bea.ide.lang.java.driver.JavaCompilerDriver JavaCompilerDriver for an example.

public boolean isFeatureAvailable(
Class featureClass)
Determines whether the given feature is in the available features list for this driver.
Parameters
featureClass - The class object for the feature in question.
Returns
true if featureClass is an available feature.

Fields **protected _doc**

Package com.bea.ide.sourceeditor.element

This package defines how the editor can be extended to support new or modified elements in the language. Elements are recognizable pieces of a document (e.g., a function name). Its classes and interfaces are described below.

public interface **com.bea.ide.sourceeditor.element.ISourceElementGroup**

Empty interface.

public interface **com.bea.ide.sourceeditor.element.ISourceElement** implements com.bea.ide.document.IDocumentElement

An ISourceElement is a specific type of IElement corresponding to the contents of a document. All parse tree elements are of type ISourceElement.

Methods **public com.bea.ide.util.Range getRange()**
Returns
the text range occupied by the source element.

public void mergeFrom(
ISourceElement sourceElement)
Merges the contents of one source element into the this source element, firing property change events as appropriate. The elements must be of compatible types. This method is used on an element-by-element basis when merging a new parse tree into an existing tree. After the merge, this.@link Object#equals equals(sourceElement) will return true while this == sourceElement will return false. Note that this method has no effect on the children of the source or destination elements: it only applies to element properties.
Parameters
sourceElement - the element from which to copy data.

public java.lang.Object getCookie()
Returns a cookie object provided by the compiler at creation time. This value has no meaning to code outside of the compiler. This method should never be called within IDE extension code.

public IStructureFeature.IAuthorNode createAuthorNode()

Used by IDE two-way editing code to create a template node that will be passed to the compiler. Note that IAuthorNodes are intended to be created, possibly modified, passed to the compiler, and then discarded. There is no guarantee that the author node will remain in the same form after being passed to the compiler.
Returns
a snapshot of the current node in a form that can be used for the purposes of two-way editing.

public IStructureFeature.StructureCode getPropertyCode(
String propertyName)
This method allows the IDE elements to maintain a property set with different names than those provided by the compiler by providing a mechanism by which IDE names can be mapped back to compiler type codes.
Parameters
propertyName - an IDE element property name
Returns
a compiler type code. The return value must be the same as that passed to the @link com.bea.ide.sourceeditor.compiler.ILanguageBuilder#setProperty setProperty call that creates properties of this type.

public com.bea.ide.sourceeditor.element.ISourceElement requestAddChild(
ISourceElement newChild)
Two way editing method allowing for the direct addition of children to an element in an ISourceElement parse tree. Once an element is part of a compiled document, modifications to the element structure must be made via 'request' methods. These methods send the modification request to the compiler, which will cause a modification of the document source code, followed by a reparse and subsequent element change events on the document. Note that this method initiates an asynchronous call; completion of the edit is indicated by an element change event, not by the return from the call.
Parameters
newChild - a template for the child element to be added. <code>newChild</code> itself will NOT be added to the parse tree, though an element equivalent to it will appear after the edit completes. Source code corresponding to this element will be inserted into the source document.
Returns
The new child added, if found. This method will return null in the case of addition failure, or in the case where the addition of a child may have caused the removal of the parent element.

public com.bea.ide.sourceeditor.element.ISourceElement requestAddChild(
ISourceElement newChild,
int index)
Two way editing method allowing for the direct addition of children to an element in an ISourceElement parse tree. Once an element is part of a compiled document, modifications to the element structure must be made via 'request' methods. These methods send the modification request to the compiler, which will cause a modification of the document source code, followed by a reparse and subsequent element change events on the document. Note that this method initiates an asynchronous call; completion of the edit is indicated by an element change event, not by the return from the call.
Parameters
newChild - a template for the child element to be added. <code>newChild</code> itself will NOT be added to the parse tree, though an element equivalent to it will appear after the edit completes. Source code corresponding to this element will be inserted into the source document.
index - The index, relative to the new child's siblings, where the element should be inserted
Returns
The new child added, if found. This method will return null in the case of addition failure, or in the case where the addition of a child may have caused the removal of the parent element.

public void requestRemoveChild(

ISourceElement child)

Two way editing method allowing for the direct removal of children from an ISourceElement parse tree. Once an element is part of a compiled document, modifications to the element structure must be made via 'request' methods. These methods send the modification request to the compiler, which will cause a modification of the document source code, followed by a reparse and subsequent element change events on the document. Note that this method initiates an asynchronous call; completion of the edit is indicated by an element change event, not by the return from the call.

Parameters

child - the element to be removed from the parse tree. All corresponding source code will be deleted from the document.

public void requestSetProperty(
String strProp,
Object value)

Two way editing method allowing for the direct manipulation of property values of an element in an ISourceElement parse tree. Once an element is part of a compiled document, modifications to the element must be made via 'request' methods. These methods send the modification request to the compiler, which will cause a modification of the document source code, followed by a reparse and subsequent element change events on the document. Note that this method initiates an asynchronous call; completion of the edit is indicated by an element change event, not by the return from the call.

Parameters

strProp - the IDE property name to be modified

value - the new value for the property. It is the caller's responsibility to ensure that the object type is compatible with property values accepted by the compiler.

public java.util.List getDiagnostics()

Returns a list of the @link com.bea.ide.sourceeditor.compiler.ICompilerDiagnostic ICompilerDiagnostic objects found in the text corresponding to this source element. Note that diagnostics overlapping the range will be returned as well as those that are completely contained.

Returns

A list of @link com.bea.ide.sourceeditor.compiler.ICompilerDiagnostic ICompilerDiagnostic objects.

public abstract class **com.bea.ide.sourceeditor.element.DefaultSourceElement** extends com.bea.ide.document.DefaultDocumentElement implements com.bea.ide.sourceeditor.element.ISourceElement

A default implementation of ISourceElement .

Constructors

public DefaultSourceElement(
Object structureNodeCookie)

Methods

public abstract IStructureFeature.StructureCode getTypeCode()

public final java.lang.Object getCookie()

public final java.beans.PropertyDescriptor[] getPropertyDescriptors()

protected abstract com.bea.ide.sourceeditor.element.DefaultSourceElement.CompilerPropertyDescriptor[ ] getDefaultPropertyDescriptors()

public IStructureFeature.IAuthorNode createAuthorNode()

```
public com.bea.ide.sourceeditor.element.ISourceElement requestAddChild(
        ISourceElement newChild)

public com.bea.ide.sourceeditor.element.ISourceElement requestAddChild(
        ISourceElement newChild,
        int index)

public void requestRemoveChild(
        ISourceElement child)

public void requestSetProperty(
        String strProp,
        Object value)

public IStructureFeature.StructureCode getPropertyCode(
        String propertyName)

public java.util.List getChildrenByClass(
        Class type)

public void addChild(
        ITreeElement child)

public com.bea.ide.sourceeditor.element.ISourceElement replaceChild(
        int index,
        ISourceElement replacement)

public com.bea.ide.sourceeditor.element.ISourceElement removeChild(
        int index)

public void removeChild(
        ITreeElement child)

public void insertChild(
        int index,
        ISourceElement newChild)

protected void initializeRange(
        IStructureFeature feature,
        ISourceModel model)
```
Initializes the range of this element and all its children. This method assumes that it is being called within a compiler read lock.

```
public com.bea.ide.util.Range getRange()

public void mergeFrom(
        ISourceElement sourceElement)

public boolean equals(
        Object o)

public java.lang.ClassLoader getClassLoader()

public void setProperty(
        String sProp,
        Object value)

public java.lang.Object getProperty(
        String sProp)

public void propertyModified(
```

String sProperty,
Object oldValue,
Object newValue)

public boolean mergeElements(
ISourceElement sourceElement)

public java.lang.String getInstanceName()

protected boolean isAuthorProperty(
DefaultSourceElement.CompilerPropertyDescriptor pd)
Allows a DefaultSourceElement derivation to specify that certain properties should not be passed to the compiler when performing a two way edit.
Parameters
pd - The property descriptor that could be passed to the compiler
Returns
boolean indicating whether the specified property should be passed.

protected java.lang.String getPropertySourceText(
IStructureFeature.StructureCode prop)

protected IStructureFeature.Range getPropertySourceRange(
IStructureFeature.StructureCode prop)

protected IStructureFeature.Range[] getPropertySourceRanges(
IStructureFeature.StructureCode prop)

public com.bea.ide.sourceeditor.element.ISourceElementGroup getGroup()
By default, elements are not grouped together (other than by parentage). To be groupable, a class must override this method.

public java.util.List getDiagnostics()

Fields **public static final PROP_NAME**

public static final PROP_VALUE

protected startPos

protected endPos

protected class **com.bea.ide.sourceeditor.element.DefaultSourceElement.DefaultAuthorElement**

Constructors **public DefaultSourceElement.DefaultAuthorElement()**

Methods **public IStructureFeature.StructureCode getTypeCode()**

public java.util.HashMap getProperties()

public java.util.List getChildren()

protected class **com.bea.ide.sourceeditor.element.DefaultSourceElement.CompilerProperty Descriptor** extends java.beans.PropertyDescriptor

Constructors

```
public DefaultSourceElement.CompilerPropertyDescriptor(
        String propertyName,
        Class beanClass,
        Object defaultValue,
        IStructureFeature.StructureCode compilerTypeCode,
        boolean displayProperty)

public DefaultSourceElement.CompilerPropertyDescriptor(
        DefaultSourceElement.StaticCompilerPropertyDescriptor pdOther)
```

Constructor which avoids the cost of doing reflection to find the getter and setter methods.

Methods

```
public java.lang.Object getDefaultValue()

public IStructureFeature.StructureCode getTypeCode()

public java.lang.String getFriendlyName()

public boolean isDisplayedProperty()

public java.lang.String getLabel()

public java.lang.Object getValue()

public java.lang.String getDescription()

public java.util.List getChildren()

public void setValue(
        Object value)
```

protected static class **com.bea.ide.sourceeditor.element.DefaultSourceElement.StaticCompilerPr opertyDescriptor** extends java.beans.PropertyDescriptor

For performance, a class that can be passed to CompilerPropertyDescriptor's constructor to save the overhead of finding the getter and setter methods.

Constructors

```
public DefaultSourceElement.StaticCompilerPropertyDescriptor(
        String strProperty,
        Class clazz,
        Object defaultValue,
        IStructureFeature.StructureCode compilerTypeCode,
        boolean displayProperty)
```

Package com.bea.ide.sourceeditor.model

This package defines ways custom editors or views can interact with the document model. The document model is a data structure that contains the text of a document. Its classes and interfaces are described below.

public interface **com.bea.ide.sourceeditor.model.ISourceModelListener**

A source model listener is notified of all changes to a source model. This includes changes to the text data as well as line attributes.

Methods

public void insertUpdate(
ISourceModelEvent e)
Called when data is inserted into the source model.

public void removeUpdate(
ISourceModelEvent e)
Called when data is removed from the source model.

public void changedUpdate(
ISourceModelEvent e)
Called when the source model changes. Line attribute modification notifications are made through this method.

public interface **com.bea.ide.sourceeditor.model.ISourceModelEvent**

A source model event indicates that the text of the source model has changed in some way. Changes include text changes and line attribute changes. See ISourceModelListener and ISourceModel.addSourceModelListener for more information.

Methods

public int getOffset()
Returns
the character offset of the event

public int getLength()
Returns
the length of change indicated by the event

public com.bea.ide.sourceeditor.model.ISourceModelEvent.EventType getType()
Returns
the type of change indicated by the event

public boolean isComplierSignificant()
Returns
boolean indicating if the change that will be passed to the compiler. Text-based changes are always compiler-significant. Changes to line attributes are not.

public static final class **com.bea.ide.sourceeditor.model.ISourceModelEvent.EventType**

The type of change indicated by the event. There are three types, each static objects that can be compared directly via references.

Methods

public java.lang.String toString()

Fields **public static final INSERT**

public static final REMOVE

public static final CHANGE

public interface **com.bea.ide.sourceeditor.model.ISourceModel**

A source model is a data structure that contains the actual text data of a given document. Generally implemented as a linked list of lines, some implementations may extend the swing AbstractDocument class, though this is not required and should not be relied upon.

Methods

public int getLineCount()
Returns
the number of lines in this source model.

public int getLineIndexByOffset(
int offset)
Parameters
offset - a zero-based character offset within the file
Returns
the zero-based index of the line containing the offset. May return null if the offset is invalid.

public com.bea.ide.sourceeditor.model.ISourceModel.Line getLineByIndex(
int index)
Parameters
index - a zero-based line index
Returns
the Line object at the specified index. May be null if the line index is invalid.

public com.bea.ide.sourceeditor.model.ISourceModel.Line getLineByOffset(
int offset)
Parameters
offset - a zero-based character offset within the file
Returns
the Line object at the specified index. May be null if the offset is invalid.

public void remove(
int offset,
int len)
Deletes text from the source model. Note that all document modifications will initiate a recompilation, which may generate element change events.
Parameters
offset - a zero-based character offset within the file
len - the length of the segment to be removed

public void insertString(
int offset,
String str,
AttributeSet a)
Inserts text into the source model. Note that all document modifications will initiate a recompilation, which may generate element change events.
Parameters
offset - a zero-based character offset within the file
str - the string to be inserted
a - a collection of attributes to place on newly created lines.

public java.lang.String getText(
int offset,
int length)

Retrieves a subset of the source model in String form.

Parameters

offset - a zero-based starting character offset within the file
length - the length of text to be returned

public void getText(
int offset,
int length,
Segment text)

Retrieves a subset of the source model in a text Segment.

Parameters

offset - a zero-based starting character offset within the file
length - the length of text to be returned
text - the Segment in which the requested text will be stored.

public java.lang.String getDenormalizedText(
int offset,
int length)

Retrieves a subset of the source model in String form. The string is converted to contain the newline characters that were found in the document text when the file was loaded. Added lines will be terminated using the first newline in the original file, or the system default value for empty files. Note that the offset and length passed in are in terms of the normalized text. The resulting string may have a different length than the length passed in.

Parameters

offset - a zero-based starting character offset within the file
length - the length of text to be extracted

public java.io.Reader getDenormalizedTextReader()

Returns a reader for the document text that denormalizes as above. This method should generally only be called when writing document contents to disk.

public int getLength()

Returns

the length, in characters, of the source model.

public void addUndoableEditListener(
UndoableEditListener l)

Adds a listener to the source model that will be notified of all undoable edits that occur.

Parameters

l - the listener to be added

public void removeUndoableEditListener(
UndoableEditListener l)

Removes an UndoableEditListener from the source model.

Parameters

l - the listener to be removed.

public void addSourceModelListener(
ISourceModelListener l)

Adds a source model listener to the source model. A source model listener is notified of all source model changes, whether they are undoable or not.

Parameters

l - the listener to be added

public void removeSourceModelListener(
ISourceModelListener l)

Removes a source model listener from the source model.

Parameters

l - the listener to remove

public void setLineAttribute(
int line,
Object key,
Object value)

Sets an attribute on the specified line. Note that modifying a line attribute value will cause the selected line to repaint, if visible in the source view. As a result, line attributes are a common means of storing the location of line specific UI information, such as breakpoint and instruction point locations.

Parameters

line - the zero-based index of the line upon which to set the attribute
key - an arbitrary key used for retrieval of the line attribute
value - line attribute value

public java.lang.Object getLineAttribute(
int line,
Object key)

Retrives the attribute with the specified key from the specified line.

Parameters

line - the zero based index of the desired line
key - an arbitrary key value, as set via a <code>@link ISourceModel#setLineAttribute setLineAttribute</code> call.

Returns

the line attribute with the specified key on the specified line. May return null is there is no such attribute on that line.

public void touchLineAttribute(
int line,
Object key)

Does not actually change a line's attributes, but does trigger a change event to be sent to listeners, as if setLineAttribute had been called. Useful when the key's value has been changed within the same object.

Parameters

line - the zero based index of the desired line
key - an arbitrary key value, as set via a <code>@link ISourceModel#setLineAttribute setLineAttribute</code> call. May be null to indicate multiple keys have been touched.

public javax.swing.text.Position createPosition(
int offs)

Creates a Position object to track a location in a document through user changes. Insertions on top of this position will move it to the end of the inserted text. Ideally, the class that implements this interface will also extend AbstractDocument, in which case you get this for free.

Parameters

offs - is the character position in the document

Returns

a right-biased Position mark.

public javax.swing.text.Position createHoldPosition(
int offs)

A variation on createPosition() which creates a Position that does not move to accomodate insertions directly on top of it. This variation is not implemented by AbstractDocument, nor supported by Swing's GapContent class. The SourceEditor's DefaultSourceModel class does implement its own GapContent to support this.

Parameters

offs - is the character position in the document

Returns

a left-biased Position mark.

public void setReadLock(

boolean on)
Acquires/releases a read lock on the model.

public void setWriteLock(
boolean on)
Acquires/releases a write lock on the model.

Fields **public static final ATTRIB_LINE_DIRTY**

public static interface **com.bea.ide.sourceeditor.model.ISourceModel.Line**

A line in the source model. It is assumed that it is safe to hold a reference to a line through document edits; Line objects must remain valid by tracking document changes and adjusting their range appropriately.

Methods **public int getStartOffset()**
Returns
the inclusive start offset of this line. Note that this may change over document edits.

public int getEndOffset()
Returns
the exclusive end offset of this line. Note that this may change over document edits.

public java.lang.String getText()
Returns
the actual text of this line. Note that this may change over document edits.

Package com.bea.ide.sourceeditor.ui

This package defines mechanisms developers can use to implement custom UI features. Its classes and interfaces are described below.

public interface **com.bea.ide.sourceeditor.ui.IPopupTip**

Methods **public java.awt.Component getComponent()**

public int getOffset()

public int getHorizontalAlignment()

public int getVerticalAlignment()

public int getTabPosition()

public boolean disposeOnMouseAction()

public boolean disposeOnKeyAction()

public void onDisplay(
IPopupTip.Container container)

public void onDispose()

Fields **public static final HORIZONTAL_LEFT_MARGIN**

public static final HORIZONTAL_AT_TOKEN

public static final HORIZONTAL_TAB_AT_OFFSET

public static final HORIZONTAL_RIGHT_MARGIN

public static final VERTICAL_BELOW_TOKEN

public static final VERTICAL_AT_TOKEN

public static final VERTICAL_ABOVE_TOKEN

public static interface **com.bea.ide.sourceeditor.ui.IPopupTip.Container**

Methods **public void relayout()**

public boolean dismiss()

public interface **com.bea.ide.sourceeditor.ui.IHoverListener**

Methods **public void hoverOccurred(**
HoverEvent hoverEvent)
Called when the mouse has hovered over a particular offset within a source view.

public class **com.bea.ide.sourceeditor.ui.HoverEvent** extends javax.swing.event.ChangeEvent

Constructors **public HoverEvent(**
ISourceView sourceView,
int offset)

Methods **public int getOffset()**
The offset of the hover.

public abstract class **com.bea.ide.sourceeditor.ui.DefaultPopupTip** implements com.bea.ide.sourceeditor.ui.IPopupTip

Constructors **public DefaultPopupTip(**
ISourceView sourceView,
int offset)

Methods **public boolean disposeOnKeyAction()**

public boolean disposeOnMouseAction()

public int getVerticalAlignment()

public int getHorizontalAlignment()

public int getTabPosition()

public int getOffset()

public void onDispose()

public void onDisplay(
IPopupTip.Container container)

Fields **protected _sourceView**

protected _offset

protected _container

Compiler Framework Client API

A compiler framework client API can define how clients, such as the editor and runtime, communicate with the compiler framework. Clients can use this API to access and modify a wide range of information about the project, its source files and object files. This API can be the mechanism by which the full compiler is integrated into the editor to provide the features described above. This section contains a complete description of all the packages that are part of the client API.

Packages	
com.bea.compiler	Defines how the editor and runtime use the compiler to get language information and produce running code.
com.bea.compiler.classfile	Defines how the client gets and sets information about the classfile
com.bea.compiler.command	Defines how the client sends commands to the compiler and receives the results.
com.bea.compiler.enterprise	Defines how the client gets additional information pertaining to enterprise applications, such as web applications.
com.bea.compiler.feature	Defines how the client gets and sets information about certain features of the souce code.
com.bea.compiler.fileinfo	Defines how the client gets and sets information about the project's files and directories.
com.bea.compiler.info	Defines how the client gets and sets information

	about the structure of a source file.
com.bea.compiler.info.beaninfo	Defines how the client gets and sets additional information available for JavaBeans
com.bea.compiler.java	Defines the information produced by the Java programming language support in the compiler.
com.bea.compiler.java.annotation	Defines the information produced by the Java programming language support sin the compiler about the annotations found in Java source code.
com.bea.compiler.typeinfo	Defines how the client gets and sets information about types.

This API defines how the editor and runtime use the compiler. The package `com.bea.compiler` contains the main interfaces. Some subpackages of this provide additional interfaces that are specific to a particular language. For example, the APIs in `com.bea.compiler.java` provide the Java-specific information for the editor and runtime.

The compiler framework is designed for integration into an editor. Like any Java compiler, the compiler framework can translate source files into Java bytecodes. But in addition, the compiler framework is designed to provide the language information that is the backbone of advanced editor features. The compiler framework makes it easy to translate the language knowledge in the compiler into editor features.

Package com.bea.compiler

This package defines how the editor and runtime use the compiler to get language information and produce running code. Its classes and interfaces are described below.

public interface **com.bea.compiler.IToken**

Contains information about a token found by a scanner.

Methods	**public boolean isOutOfDate()** Returns whether this token is known to have been changed since the last compile.
	public int getType() Returns the type code of this token.
	public int getStart() Returns the inclusive start of the range of this token.
	public int getEnd() Returns the exclusive end of the range of this token.
	public int getSize() Returns the size of the range of this token.
Fields	**public static final UNKNOWN** The token type for an 'unknown' token.

public interface **com.bea.compiler.ITextChange**

Represents a change that occurred in the text of a source file.

Methods	**public int getType()** Returns the type of change that occurred.
	public int getOffset() Returns the offset at which the change occurred.
	public int getLength() Returns the length of the change.
Fields	**public static final INSERTED** Indicates that text was inserted.
	public static final REMOVED Indicates that text was removed.
	public static final REPLACED Indicates that text was replaced.

public interface **com.bea.compiler.IProjectChangeListener**

Defines the shape of project change notifcations to the client.

Methods	**public void changesFound(IProjectChangeEvent event)** Provides the metadata that was found to have been changed from changes previously delivered by applyChanges .

public interface **com.bea.compiler.IProjectChangeEvent**

Encapsulates all of the changes found during a compilation.

Methods	**public java.util.Collection getChanges()** Returns the IFileChange s that were applied in this compile or null if this change did not correspond to any changes.
	public java.util.Collection getCompiledFiles() Returns the IFile s that were recompiled.
	public java.util.Collection getChangedFiles() Returns the IFile s whose changes were picked up during this compile.
	public java.util.Collection getOutOfDateFiles() Returns the IFile s that were not change but were found to be out of date during compilation.
	public java.util.Collection getDiscoveredFiles() Returns the source IFile s that were found on the classpath during compilation.

public java.util.Collection getInfoChanges(
int type,
String namespace)
Returns all of the changes of the given type(s) in the given namespace (or all namespaces if null) that were found duringn compilation.

public interface **com.bea.compiler.IProject**

Maintains compiler information about all of the source files in a project. This information is automatically updated when notified of changes to the source files. When requested, it will produce a .class files.

Note that the project does not take a source path argument. Instead, the source path is managed by applying changes to the project. Thus, to add files to a new project, the client must apply a change list containing an add for each of the files.

Methods

public void setBinaryPaths(
URI[] paths)
Sets the paths containing .class files.

public java.net.URI[] getBinaryPaths()
Returns the paths containing .class files.

public com.bea.compiler.IFileStore.IRoot[] getBinaryRoots()
Returns the roots for the binary path.

public void setSourcePaths(
URI[] paths)
Sets the paths containing source files. These will only be used if a class is referenced that cannot be found in files given directly.

public java.net.URI[] getSourcePaths()
Returns the paths containing source files.

public com.bea.compiler.IFileStore.IRoot[] getSourceRoots()
Returns the roots for the source path.

public void setOutputDir(
URI dir)
Sets the directory into which classes are placed. The URI given is assumed to be a heirarchical, so that appending class files can be placed into sub-directories of the one given.

public java.net.URI getOutputDir()
Returns the directory into which classes are placed.

public void setOutputDirector(
IProject.IOutputDirector dir)
Sets a director for this project. This overrides the output dir.

public void setInUseFile(
IProject.IInUseFile iinuse)
Sets the IInUseFile interface that will be used by the project when a file is added. By default, all files start out by not being in use

public void ignoreDependencies(
URI[] roots)
Sets the roots on which dependencies are ignored. By default, all binary roots are ignored.

public com.bea.compiler.IFile getSourceFile(

URI path)
Returns source file information for the given file.

public java.util.Collection getSourceFiles()
Returns a Collection of URIs of all source files known to the project. getSourceFile() can be called on each URI in the collection.

public com.bea.compiler.INamespace getNamespace(
String name)
Returns the namespace with the given name.

public com.bea.compiler.info.IPackage getPackageRoot()
Returns the root package object.

public com.bea.compiler.info.IName getName(
String name)
Returns a name object for the given qualified name.

public com.bea.compiler.info.IPackageMember getEntity(
IName name)
Returns the package or type with the given name.

public com.bea.compiler.info.IPackage getPackage(
IName name)
Returns the package with the given name.

public com.bea.compiler.info.IReferenceType getUserType(
IName name)
Returns the class or interface with the given name.

public com.bea.compiler.fileinfo.IFileInfo getFileInfoRoot()
Returns the node for the given file.

public void addChangeListener(
IProjectChangeListener listener)
Adds the given change listener to this project.

public void removeChangeListener(
IProjectChangeListener listener)
Removes the given change listener from this project.

public void applyChanges(
Collection changes,
boolean toBuild)
Notifies the compiler that the files with the given URIs have changed. The compiler will shortly update the source information and notify all listeners of the change.

Note that, although every apply will be followed by a change notification, the number of applies may be greater than the number of notifications because the compiler can choose to batch multiple applies together.
Parameters
changes - A collection of <code>IFileChange</code> objects.
toBuild - Optimize for a subsequent build

public void build(
Collection files,
boolean force)
Attempts to generate .class files for all of the given source files plus their dependencies. This will omit any files whose dependencies (including itself) contain errors.
Parameters
files - Provides the <code>URI</code> paths of each of the files to be code-generated. If this is null, then all known source files will be generated.

force - Indicates that all of the given files should be generated. Otherwise, this will not generate files (1) if they have not changed (including dependencies) since their last generation or (2) if this would overwrite class files that are newer than any of the changes (including dependencies).

public void acquireReadLock()
Takes a read lock on the project. No compilation will occcur until this lock is released.

public void releaseReadLock()
Releases the read lock on the project.

public com.bea.compiler.IDiagnosticSet getDiagnostics(
Collection files)
Returns the set of diagnostics in the given files. Each entry in the collection should be the URI path of the file.

public com.bea.compiler.IDiagnosticSet getDiagnostics(
URI file,
boolean includeDeps)
Returns the set of diagnostics in the given file and, if requested, all of it's dependencies.

public java.lang.Object getProperty(
String name,
Object defValue)
Returns the value of the given property on this project or the given default value if it has not been set.

public void setProperty(
String name,
Object value)
Sets the given property on this project.

public com.bea.compiler.feature.IFeature createFeature(
Class key)
Returns the feature with the specified key. Project features have no state and hence do not need to be destroyed.

public void close()
Drops any outstanding resources held by this project.

Fields

public static final PROP_KEEP_BUFFERS
If set to Boolean.TRUE, this property will stop the compiler from releasing the buffers in each file compiled.

public static final PROP_NO_FEATURES
If set to Boolean.TRUE, features will not be supported, but the project can then drop many data structures after a file is compiled.

public static final PROP_CLASSLOADER
If the compiler is not running with the project jars in the default ClassLoader , as it does on the server, then the client should set this property with a ClassLoader for project jars to allow languages to run code from the user's project. (e.g control proprety validation)

public static interface **com.bea.compiler.IProject.IOutputDirector**

Provides a mapping from source file to a root under which all the generated classes for the file will be placed.

Methods **public com.bea.compiler.IFileStore.IRoot[] getRoots()**
Returns all of the possible roots to which classes could be sent.

**public com.bea.compiler.IFileStore.IRoot getRoot(
IFile file)**
Returns the root for classes generated from the given file.

public static interface **com.bea.compiler.IProject.IInUseFile**

Methods **public boolean isInUse(
URI path)**
Returns true if the file represented by this URI should start out as in use

public interface **com.bea.compiler.INode**

Represents a node in the tree of nested languages in a source file.

Methods **public com.bea.compiler.ILanguage getLanguage()**
Returns the language in this node.

public int getStart()
Returns the inclusive start of the range of text for this node.

public int getEnd()
Returns the exclusive end of the range of text for this node.

public java.util.Iterator getChildren()
Returns the children of this node.

public interface **com.bea.compiler.INamespace**

Maintains information about some sort of objects that can be defined by source files in a project.

Methods **public java.lang.String getName()**
Returns the name of this namespace. To avoid collisions, domain-qualified names should be used.

public static interface **com.bea.compiler.INamespace.IInfoChange**

Represents a change that was found during compilation regarding an object in this namespace.

Methods **public java.lang.String getNamespace()**
Returns the name of the namespace that generated this change. This will typically imply a subclass of IInfoChange to which this can be cast.

public int getType()

Returns the type of change that occurred.

Fields	**public static final TYPE_ADDED** Indicates that an object was added. **public static final TYPE_REMOVED** Indicates that an object was removed. **public static final TYPE_MODIFIED** Indicates that an object was modified.

public interface **com.bea.compiler.ILanguage**

Represents a particular language supported by the compiler.

Methods	**public java.lang.String getName()** Returns a unique short name for this language. This should be the first part of codes used in diagnostics.

public interface **com.bea.compiler.IFileStore**

Encapsulates knowledge about how to navigate a particular type of store for source files. New file stores can be registered with the compiler.

Methods	**public java.lang.String getScheme()** Returns the name that is used to identify this file store in the scheme part of the URI. **public com.bea.compiler.IFileStore.IRoot getRoot(URI root)** Returns an object for looking up files under this root. **public com.bea.compiler.IFileStore.IEntry getEntry(URI file)** Returns an entry with an anonymous root if the given file exists. This will not have much useful information to provide about the relative path inside the root. **public java.lang.String getExtension(URI file)** Returns the extension of the source file. This is used to determine which language is contained in the file.

public static interface **com.bea.compiler.IFileStore.IRoot**

Represents the root of a tree of files. This allows lookup for specific files under that root.

Methods	**public com.bea.compiler.IFileStore.IEntry getEntry(String name)** Returns an entry for the file with the given ('/' separated) name under this root or null if none exists. Note that this path should not include any URI-encoded characters. **public com.bea.compiler.IFileStore.IEntry getEntry(**

String rootName,
String[] extensions)
Returns an entry for a root name, with a list of possible extensions. Will also return a directory/package entry if none of the extensions exist, but a directory does.

public com.bea.compiler.IFileStore.IEntry createEntry(
String name)
Returns an entry that may not exist but can be written to.

public java.util.Iterator getEntries()
Returns an iterator over all of the entries under this root.

public java.util.Iterator getEntries(
String dir,
String[] extensions)
Returns an iterator over all of the entries under this root in the given directory and with one of the given extensions or with no extension, if it is a directory.

public boolean contains(
URI file)
Determines whether the given file is in this root.

public java.lang.String getRelativePath(
URI file)
Returns the relative path to the given file in this root, without any URI-encoded characters.

public java.net.URI getPath()
Returns the path to this root.

public java.lang.String getSystemPath()
Returns the system path to this root.

public void close()
Drops any outstanding resources held by this root.

public boolean canContainSource()
Determines whether this root can contain source files.

public static interface **com.bea.compiler.IFileStore.IEntry**

Represents an entry (file or directory) under a particular root.

Methods

public java.net.URI getPath()
Returns the absolute URI for this entry.

public java.lang.String getRelativePath()
Returns the relative path for this entry inside the root.

public java.lang.String getName()
Returns the name of this entry.

public java.lang.String getExtension()
Returns the extension of this entry.

public java.lang.String getSimpleName()
Returns the name without any extension.

public java.lang.String getSimplePath()

Returns the relative path without any extension.

public boolean isDirectory()
Determines whether this entry represents a directory or a file.

public boolean exists()
Determines whether this entry really exists. This is only useful \for entries returned by createEntry .

public com.bea.compiler.IFileStore.IRoot getRoot()
Returns the root that created this entry.

**public java.io.InputStream getInputStream(
long offset)**
Returns an input stream for this file at the given offset.

Note that during the compilation of source files with nested languages (almost every file), multiple input streams will be requested on the same file. Implemenations may wish to optimize for this, possibly by reading into a buffer on the first request.

public java.io.OutputStream getOutputStream()
Returns an output stream for this file.

public java.lang.String getSystemPath()
Returns a path to this file on the system (i.e. an argument that could be passed to the constructor of java.io.File). If the * store does not support this operation, it may throw an exception.

public java.lang.String getDisplayPath()
Returns a path which can be used for printing diagnostics.

public long getLastModified()
Returns the last modified time of the file on disk.

public interface **com.bea.compiler.IFileChange**

Represents a change that was found in a source file of the project. These objects will be used to notify the compiler of the changes.

Methods

public java.net.URI getPath()
Returns the path of the file that changed.

public int getType()
Returns the type of change that occured.

public java.util.Collection getTextChanges()
Returns a list of the ITextChange s for this modification or null if this was not a modification. If there are multiple changes in the collection, they must be ordered as they occurred in the time. Furthermore, the offsets in each change will refer to the state of the world after the previous changes are applied.

public long getBufferVersion()
If this is a modification, then this records the version of the buffer for this file to which these change apply (in order to get to the next version). If a pending compilation has just retrieved a buffer for this file and its version is greater than this, then it will ignore this change.

public boolean isInMemory()
Determines whether this represents a change to the file in-memory or on disk. This affects

how the last-modified stamp on the file is updated.

Fields

public static final ADDED
Indicates that the file was added.

public static final REMOVED
Indicates that the file was removed.

public static final MODIFIED
Indicates that the file was modified.

public interface **com.bea.compiler.IFile**

Encapsulates information about a file. The file may contain source text or it may be a file/jar of compiled class(es).

Methods

public com.bea.compiler.IProject getProject()
Returns the project containing this file.

public com.bea.compiler.IFileStore.IEntry getEntry()
Returns the entry for this file.

public java.net.URI getPath()
Returns the path of this source file.

public boolean isSource()
Determines whether this is a source rather than a binary file.

public boolean isBinary()
Determines whether this is a binary rather than a source file.

public boolean foundOnPath()
Determines whether this file was found on the source or binary path. If this is not the case, then the file was added to the project directly. If this is the case, then the file was discovered during compilation, and was not added directly.

public boolean isRemoved()
Determines whether this file has been removed from the project.

public java.util.Iterator getTypes()
Returns the top-level types declared in this file.

public com.bea.compiler.fileinfo.IFileInfo getFileInfo()
Returns the record for this file in the file namespace.

public com.bea.compiler.feature.IFeature createFeature(
Class key)
Returns the feature with the specified key. The file cannot be minimized until all features have been destroyed.

public void destroyFeature(
IFeature feature)
Cleans up a feature. Every feature created with createFeature should be cleaned up with a call to this function.

public void setInUse(
boolean inUse)
Marks a file as not being in use, which means no features will be requested on it

public java.util.Iterator getDependencies()
Returns an iterator over all files that this file is dependant on, including itself. This file will always be the first in the list, so it can be skipped with one call to next() .

Each entry in the list will be an IFile .

public interface **com.bea.compiler.IDiagnosticSet**

Encapsulates a set of diagnostics found in the source. The encapsulated set of diagnostics is grouped into entries, where each entry corresponds to a file that contains some diagnostics.

Methods

public int size()
Returns the total number of diagnostics found.

public int getErrorCount()
Returns the number of error diagnostics found.

public int getWarningCount()
Returns the number of warning diagnostics found.

public int getInfoCount()
Returns the number of informational diagnostics found.

public java.util.Iterator entries()
Returns an iterator over the entries for each file that contains diagnostics. The order of these entries will match the order in which the files were given when this set was created.

public java.util.Iterator iterator()
Returns an iterator over all of the diagnostics found. These will be ordered by file and then location in the file.

public void print(
PrintWriter output)
Outputs a description of the diagnostics to the given writer.

public void print(
PrintStream output)
Outputs a description of the diagnostics to the given stream.

public static interface **com.bea.compiler.IDiagnosticSet.IEntry**

Encapsulates a file all of of its diagnostics.

Methods

public com.bea.compiler.IFile getFile()
Returns the file that contains diagnostics.

public java.util.Iterator getDiagnostics()
Returns the diagnostics in this file sorted by the position at which they apply.

public interface **com.bea.compiler.IDiagnostic**

Contains information about an error detected in the source code. Note that the range returned is inclusive on both ends.

Note that a 'column' in this context is a count of the number of characters. Specifically, a tab increases the column count by one.

Methods

public com.bea.compiler.IFile getFile()
Returns the file in which this diagnostic was reported.

public int getType()
Returns the type of this diagnostic.

public java.lang.String getDescription()
Returns the description of the diagnostic.

public java.lang.String getPrescription()
Returns the prescription for the diagnostic or null if none is exists.

public int getFirstLine()
Returns the line on which this diagnostic starts.

public int getFirstColumn()
Returns the column at which this diagnostic starts.

public int getLastLine()
Returns the line on which this diagnostic ends.

public int getLastColumn()
Returns the column at which this diagnostic ends.

public java.lang.String getCode()
Returns a code for the error. This should never be shown to a user. It is provided for testing and debugging purposes only.

Fields

public static final TYPE_ERROR
Indicates a problem that prevents generating code.

public static final TYPE_WARNING
Indicates a problem that does not prevent generating code.

public static final TYPE_INFO
Indicates information that is not necessarily a problem.

public static final TYPE_IGNORED
Indictes that this diagnostic will never be returned to the client.

public interface **com.bea.compiler.ICompiler**

Maintains global information about the configuration of the compiler. Multiple projects may be outstanding at the same time, all of which share this configuration.

Methods

public void configure()
Applies the default configuration to the compiler.

public void configure(
InputStream input)
Reads configuration information from the XML in the given stream.

public void configure(
Element elem)
Reads configuration information from an XML fragment.

public com.bea.compiler.IProject createProject()
Returns an empty project to which source files can be added. It is the responsibility of the caller to manage this object. It will not be collected until all references to it are removed and all outstanding updates are complete.

public void addFileStore(
IFileStore store)
Registers the given file store with the compiler. This store will be queried for any URIs that begin with its scheme.

public com.bea.compiler.IFileStore getFileStore(
String scheme)
Returns the registered store with the given scheme.

public void addBufferCache(
IBufferCache cache)
Registers the given buffer cache with the compiler. This cache will be queuried when a new buffer is needed.

public void removeBufferCache(
IBufferCache cache)
Unregisters the given buffer cache with the compiler.

public void addNamespace(
INamespace ns)
Registers the given namespace with the compiler. This namespace will then be instantiated for every project that is created thereafter.

This language must implement the namespace interfaces defined in the the compiler framework Language API .

public com.bea.compiler.INamespace getNamespace(
String name)
Returns the prototype instance of the namespace with the given name.

public java.util.Iterator getNamespaces()
Returns the prototypes of all registered namespaces.

public void addLanguage(
Class type,
ILanguage language)
Registers the given language with the compiler. This should be called once for each language before any calls are made to addExtension .

The first parameter is a class that can be used as a unique key for this language. Typically, this will be language.getClass .

This language must implement the language interfaces defined in the the compiler framework Language API .

public com.bea.compiler.ILanguage getLanguage(
Class type)
Returns the instance of the the given language type.

public java.util.Iterator getLanguages()
Returns all of the languages registered with the compiler.

public void addExtension(
String extension,
ILanguage language,
int priority)

Registers the language that will be used for compiling files with the given extension. This language should have been registered previously with a call to addLanguage . The priority is used (if this language supports discovery) to choose between multiple files that would define the same type when discovered on the source path.

public com.bea.compiler.ILanguage getExtension(
String extension)

Returns the language registered for this extension or null if no language has been registered.

public void setDefaultLanguage(
ILanguage language)

Sets the language to use when no registered extension matches.

public com.bea.compiler.ILanguage getDefaultLanguage()

Returns the language used when no registered extension matches.

public void addProjectFeature(
Class clazz,
Object feature)

Registers a new project feature with the compiler. The first parameter must be the class object for the feature interface. The second parameter must implement the interface defined in the the compiler framework Language API .

public void addFileFeature(
Class clazz,
Object feature)

Registers a new file feature with the compiler. The first parameter must be the class object for the feature interface. The second parameter must implement the interface defined in the the compiler framework Language API .

public void addBackEnd(
Class clazz,
IBackEnd backEnd)

Sets the default back-end for all languages. This back-end must implement the compiler interfaces defined in the the compiler framework Language API .

public java.lang.Object getProperty(
String name,
Object defValue)

Returns a global property of the compiler or the given default value if the property has not been set.

public void setProperty(
String name,
Object value)

Sets a global property of the compiler.

public void setProperty(
String name,
String value,
String type)

Sets a global property of the compiler. The given value is tumed into the appropriate object for its type (boolean, integer, or string).

public void addDescriptions(
ResourceBundle bundle)

Reads diagnostic descriptions from the given resource bundle. The keys in the bundle should be the codes for diagnostics and the values are their descriptions.

public void addPrescriptions(
ResourceBundle bundle)
Reads diagnostic prescriptions from the given resource bundle. The keys in the bundle should be the codes for diagnostics and the values are their descriptions.

public void addSeverities(
ResourceBundle bundle)
Reads diagnostic severities from the given resource bundle. The keys in the bundle should be the codes for the diagnostics and the values are either 'fatal-error', 'error', 'warning', 'info', and 'ignored'. The 'fatal-error' value means the same as 'error' except that it denotes the diagnostic cannot be reassigned any other severity.

public void addExceptionListener(
ICompiler.IExceptionListener listener)
Adds a listener for uncaught exceptions that occur in the compiler.

public void removeExceptionListener(
ICompiler.IExceptionListener listener)
Removes a listener for uncaught exceptions that occur in the compiler.

public void error(
int type,
String code,
Object[] args)
Outputs the given error to the compiler's error stream.

public void error(
PrintStream out,
PrintStream err,
int type,
String code,
Object[] args)
Outputs the given error to the given output or error stream.

public void output(
String code,
Object[] args)
Outputs the given error to the compiler's output stream. This is similar to an information message, but this guarantees that no prefix or suffix will be added to the message.

public void output(
PrintStream out,
String code,
Object[] args)
Outputs the given error to the given output stream.

public java.lang.String getMessage(
String code,
Object[] args)
Returns the message without writing it.

Fields

public static final PROP_DEFAULT_BACKEND
Names the property that defines the default backend. Its value should be an Object .

public static final PROP_ENCODING
Names the property that defines the default encoding for source files given to the compiler. Its value should be a String .

public static final PROP_DELAY
Names the property that defines the amount of time to wait after a change occurs before recompiling. Its value should be an Integer .

public static final PROP_VERBOSE
Names the property that determines which calls to error should be output to stdout and stderr. Its value is an Integer . All severities less than or equal to this number will be output.

public static final PROP_DEBUG
Names the property that determines whether to emit debugging information into .class files. Its value should be a Boolean .

public static final PROP_PRODUCTION
Names the property that determines whether to emit code for production mode (rather than development mode). Its value should be a Boolean .

public static final PROP_PATH_CACHE_SIZE
Names the property that determines the maximum number of entries allowed in the compiler projects' path cache. Its value should be an Integer . Default value is 4096.

public static final FATAL
Indicates an error for which we cannot continue. This will cause a System.exit.

public static final ERROR
Indicates an error for which we can continue.

public static final WARNING
Indicates a warning message.

public static final INFO
Indicates an informational message.

public static interface **com.bea.compiler.ICompiler.IExceptionListener**

Defines the events provided to listeners for uncaught exceptions.

Methods

public void uncaughtException(
Thread t,
Throwable e)
Indicates that an uncaught exception occurred on the given thread.

public interface **com.bea.compiler.IBufferCache**

Provides instances of IBuffer to the compiler if they exist in the cache.

Methods

public com.bea.compiler.IBuffer getBuffer(
URI path,
long changeID)
Returns the cached buffer for the given path or null if none exists. The buffer will contain a of the contents of the file as of the given changeID if the ID is nonnegative. Otherwise, it will be one of the constants defined below.

It is safe to assume that the compiler will never use more than two buffers for the same file at a time, and if two are used, one of those will be the "online" buffer.

Fields

public static final CURRENT_ONLINE
Indicates to return a buffer that contains the current state. But this will only used to make updates to the token set in an online fashion, i.e., in the call to applyChanges . For that

reason, it is safe to return the buffer in the editor.

public static final CURRENT_OFFLINE
Indicates to return a buffer that contains the current state. This will be used when updates are being made in an offline fashion, i.e., on a background compiler thread.

public interface **com.bea.compiler.IBuffer**

Encapsulates access to the contents of a source file. The file in question can be in memory or on disk.

Note that the contents of the buffer should never change asynchronously to the compiler. In other words, a buffer should represent a of the source code, and unless the compiler requests a change to the text by calling replaceText , the contents should not change.

Methods

public long getVersion()
Returns the version number for this snapshot of the source. The only requirements on this number are that it only increase over time.

public int getLength()
Returns the current length of the buffer.

public java.io.Reader getReader(
int start,
int end)
Returns a reader that will return the characters in the given range
Parameters
start - The inclusive start of the range to read.
end - The exclusive end of the range to read or -1 for the end of the file.

public java.lang.String getText(
int start,
int end)
Returns the text in the given range. This can only be called after call to getReader .
Futhermore, it should only be used for small text ranges.

public void replaceText(
String text,
int start,
int end)
Replaces the text in the specified range with the specified text.
Parameters
text - the text to insert
start - the start offset of the text to replace
end - the end offset of the text to remove

public int[] getLineOffsets()
Returns an array containing start offset of each line in the source.

public interface **com.bea.compiler.IBackEnd**

Represents a particular back-end supported by the compiler.

Methods

public java.lang.String getName()
Returns a unique name for this back-end.

public class **com.bea.compiler.Compiler**

Stores a reference to the single instance of the class that implements ICompiler .

Constructors **public Compiler()**

Methods **public static com.bea.compiler.ICompiler get()**
Returns a reference to the implemention of ICompiler .

public static void set(
ICompiler compiler)
Sets the reference to the implementation of ICompiler . This will be called once when the implementation class is loaded.

Package com.bea.compiler.classfile

This package defines how the editor and runtime use the compiler to get language information and produce running code. Its classes and interfaces are described below.

public class **com.bea.compiler.classfile.ClassFileAnnotation** implements java.io.Serializable

This class defines the shape of annotation entries put into the .class files generated by the compiler. The value of each attribute is a serialized version of an array of these ClassFileAnnotation .

Constructors **public ClassFileAnnotation()**

public ClassFileAnnotation(
String tagName)

Methods **public java.lang.String getName()**
Returns the name of this annotation.

public void setName(
String name)
Stores the name of this annotation.

public java.lang.String[] getAttributeNames()
Returns the names of the attributes on this annotation.

public java.lang.String getAttributeNames(
int index)
Returns the name of the attribute with the given index.

public void setAttributeNames(
String[] attrNames)
Stores the names of the attributes on this annotation.

public void setAttributeNames(
int index,
String attrName)
Stores the name of the attribute with the given index.

public java.lang.Object[] getAttributeValues()
Returns the values of the attributes on this annotation.

public java.lang.Object getAttributeValues(
int index)
Returns the value of the attribute with the given index.

public void setAttributeValues(
Object[] attrValues)
Stores the values of the attributes on this annotation.

public void setAttributeValues(
int index,
Object attrValue)
Stores the value of the attribute with the given index.

public void addAttribute(
String name,
Object value)

public java.lang.String toString()

public static byte[] toByteArray(
ClassFileAnnotation[] annotations)
Converts the given annotation array into a byte stream as would be stored into the .class file.

public static com.bea.compiler.classfile.ClassFileAnnotation[] fromByteArray(
byte[] bytes)
Converts a byte stream as would be found in a .class file into an ClassFileAnnotation array.

Fields

public static final ATTRIBUTE_NAME
Defines the name of the VM attribute containing the annotations.

Package com.bea.compiler.command

This package defines how the client sends commands to the compiler and receives the results. Its classes and interfaces are described below

public interface **com.bea.compiler.command.ICmdCompiler**

Methods

public com.bea.compiler.command.ICmdCompiler.IResult exec(
String[] args)
Performs the compilation described by the given command-line args.

public com.bea.compiler.command.ICmdCompiler.IResult exec(
String[] args,
PrintStream out,
PrintStream err)
Performs the compilation described by the given command-line args, with output sent to the given streams.

public java.lang.Object clone()
Returns a new instance of the command-line compiler.

public static interface **com.bea.compiler.command.ICmdCompiler.IResult**

Records information about the result of compilation.

Methods	**public boolean isSuccess()** Determines whether no errors were found (and code was produced). **public com.bea.compiler.IDiagnosticSet getDiagnostics()** Returns the set of errors and warnings found in the source. This may be null if no errors were found.

public class **com.bea.compiler.command.CmdCompiler**

Provides the ability to instantiate command-line compilers.

Constructors	**public CmdCompiler()**
Methods	**public static com.bea.compiler.command.ICmdCompiler.IResult exec(String[] args)** Runs the command-line compiler with the given arguments. **public static com.bea.compiler.command.ICmdCompiler.IResult exec(String[] args, PrintStream out, PrintStream err)** Runs the command-line compiler with the given arguments and streams. **public static void set(ICmdCompiler prototype)** Sets the prototype instance of the command-line compiler.

Package com.bea.compiler.enterprise

This package defines how the client gets additional information pertaining to enterprise applications, such as web applications. Its classes and interfaces are described below

public interface **com.bea.compiler.enterprise.IEntProject**

Defines additional information needed for an enterprise project. By "enterprise", we mean both J2EE and the WebLogic Workshop framework. For example, this information may be provided in web.xml.

Methods	**public com.bea.compiler.enterprise.IEntProject.IWebAppProject getWebAppProject()** Returns a web app project, or null, if this project is not a web app. **public com.bea.compiler.enterprise.IEntProject.IWebAppProject[] getWebAppProjects()** Returns all web app projects that are part of the current application.
Fields	**public static final PROP** The key for storing this on an IProject .

public static interface **com.bea.compiler.enterprise.IEntProject.IWebAppProject**

Exposes the information available on a enterprise project that is a web application project.

Methods

public java.net.URI getWebRootPath()
Returns the web root path for use in dealing with URL references.

public java.lang.String getWebContextPath()
Returns the context path for this project. A URL to a resource in this project is composed of: http://<server>:<;port>/<;context-path>/<;source-path> Thus it is necessary that the compiler know the context path in order to make any sense of a 'root-relative' URL: <a href="/samples/jsp/HelloWorld.jsp">Hello<;a> In the above case the context path is '/samples'.

Package com.bea.compiler.feature

This package defines how the client accesses information about certain features of the souce code. Its classes and interfaces are described below.

public interface **com.bea.compiler.feature.ITokenInfoFeature** implements com.bea.compiler.feature.IFeature

Methods

public com.bea.compiler.IToken getToken(
INode node,
int offset)
Returns the token at the given offset.

public java.util.Iterator getTokens(
INode node,
int offset)
Returns an iterator starting with the token at the given offset and ending at the end of the language node.

public java.util.Iterator getTokens(
INode node,
int offset,
boolean forward)
Returns an iterator starting with the token at the given offset, ending at the end of the the language node, and moving in the given direction.

public java.util.Iterator getTokens(
INode node,
int offset,
int length,
boolean forward)
Returns an iterator starting with the token at the given offset, moving in the given direction, and ending after the given length or the end of the language node, whichever comes first.

public java.lang.String getTokenName(
INode node,
IToken token)

Returns a displayable name for the specified token.

public java.lang.String[] getTokenNames(
INode node)
Returns all the token names for the language in the given node.

public java.lang.String[] getTokenNames(
ILanguage lang)
Returns all the token names for the given language.

public interface **com.bea.compiler.feature.IStructureFeature** implements com.bea.compiler.feature.IFeature

Used to build structure trees, which are client accessible representations of the parse tree. Structure trees are medium used for two-way editing.

Methods

public com.bea.compiler.feature.IStructureFeature.IStructureNode createStructure(
IStructureFeature.IStructureBuilder builder)
Creates a new structure tree using the specified builder.

public com.bea.compiler.feature.IStructureFeature.Range getRangeInSource(
IStructureFeature.IStructureNode node)
Returns the range of the specified node in the source code.

public com.bea.compiler.feature.IStructureFeature.Range getRangeInSource(
IStructureFeature.IStructureNode node,
IStructureFeature.StructureCode propCode)
Returns the range of the specified property in the source code.

public java.lang.Object getSourceValue(
IStructureFeature.IStructureNode node,
IStructureFeature.StructureCode propCode)
Returns the syntactic value of the specified property.

public void addChild(
IStructureFeature.IStructureNode parent,
IStructureFeature.IAuthorNode child)
Adds the specified child to the specified parent. This method is used to two-way edit a particular node.

public void removeChild(
IStructureFeature.IStructureNode child)
Removes the specified child from the specified parent. This is used to two-way edit a particular node.

public void replaceNode(
IStructureFeature.IStructureNode oldNode,
IStructureFeature.IAuthorNode newNode)
Replaces a particular structure node with a new node. This is used to two-weay edit a particular node.

public static class **com.bea.compiler.feature.IStructureFeature.Range**

Stores a range in the source code.

Constructors **public IStructureFeature.Range(**
int start,
int end)

Fields **public start**
The start of the range.

public end
The end of the range.

public static interface **com.bea.compiler.feature.IStructureFeature.IStructureBuilder**

An interface used to build structure trees. An IStructureBuilder must be implemented by the user of this feature and passed to createStructure . The builder will be called in order to build the structure tree.

Methods **public com.bea.compiler.feature.IStructureFeature.IStructureNode createNode(**
IStructureFeature.StructureCode typeCode,
Object cookie)
Provides the builder with a chance to create a node.

public void setProperty(
IStructureFeature.IStructureNode node,
IStructureFeature.StructureCode propCode,
Object value)
Tells the builder to set a particular property on the node.

public void addChild(
IStructureFeature.IStructureNode node,
IStructureFeature.IStructureNode child)
Tells the builder to add a child to a node.

public static interface **com.bea.compiler.feature.IStructureFeature.IStructureNode**

An interface that must be implemented by the nodes in a structure tree.

Methods **public com.bea.compiler.feature.IStructureFeature.StructureCode getTypeCode()**
Returns the type code for this node.

public java.lang.Object getCookie()
Returns the cookie for this node.

public static interface **com.bea.compiler.feature.IStructureFeature.IAuthorNode**

An interface that must be implemented by nodes that are used to author code.

Methods **public com.bea.compiler.feature.IStructureFeature.StructureCode getTypeCode()**

Returns the type code for this node.

public java.util.HashMap getProperties()
Returns the properties for this node. The keys should all be instances of StructureCode .

public java.util.List getChildren()
Returns the children for this node.

Fields

public static final INDEX
Indicates where the new node should be placed inside the parent.

public static class **com.bea.compiler.feature.IStructureFeature.StructureCode** implements java.lang.Comparable

Represents a "code" used is used to identify a type of nodes or a property on a node. Each code can be thought of as a pair of the form (class, code). This allows each language to define its own space of codes, which cannot conflict with another language's codes.

Constructors

protected IStructureFeature.StructureCode(
int code)

Methods

public int getCode()
Returns the class-specific type code.

public boolean equals(
Object o)

public int hashCode()

public java.lang.String toString()

public int compareTo(
Object o)

public interface **com.bea.compiler.feature.ISourceInfoFeature** implements com.bea.compiler.feature.IFeature

This is similar to the ITokenInfoFeature but provides more general information about the source. For that reason, it can be used only on files that have completed the front-end of compilation.

Methods

public com.bea.compiler.feature.ISourceInfoFeature.Range getRange(
int offset,
Object obj)
Returns the start offset of the given object, which was returned from querying about the given offset, or null if it is not known.

public java.lang.Object getIdentifierInfo(
int offset)
Returns the object referenced by the identifier at the given offset or null if the information could not be produced.

public boolean isCompletionChar(
int offset,

char ch)
Determines whether the given character, just typed, should initiate an attempt at completion (after a pause).

**public com.bea.compiler.feature.ISourceInfoFeature.Range getCompletionRange(
int offset)**
Returns the range in the text that holds the current value of the completion that has been chosen. This is what will be replaced when a new completion is chosen.

**public java.util.List getCompletions(
int offset,
char ch)**
Returns all of the possible objects whose names could be used in place of the identifier at the given offset (or that would be at the given offset). The return value will be null if the information could not be produced.

**public java.util.List getEnclosingContext(
int offset)**
Returns information about the enclosing contexts of the given offset. The closest context will be first in the list.

public static class **com.bea.compiler.feature.ISourceInfoFeature.Range**

Records a range of offsets in the source of the form [start,end).

Constructors	**public ISourceInfoFeature.Range()**
Fields	**public start**
	public end

public static class **com.bea.compiler.feature.ISourceInfoFeature.MethodCallContext**

Provides enclosing context for a method call.

Constructors	**public ISourceInfoFeature.MethodCallContext()**
Fields	**public location** The type in which the method would need to be located.
	public name The name of the method being called.
	public position Returns the position of the identifier part of the method call.
	public method The method currently being called (if a match exists).

public interface **com.bea.compiler.feature.IPropertyInfoFeature** implements

com.bea.compiler.feature.IFeature

Provides information about the properties of a certain strucutre node, including the valid tags and their attributes.

Methods

**public com.bea.compiler.feature.IPropertyInfoFeature.IDescription getDescription(
        IStructureFeature.IStructureNode node)**
Returns the description associated with the given node

**public java.util.Iterator getAllTags(
        IStructureFeature.IStructureNode node)**
Returns the list of possible annotation tags on the given node. The iterator elements must be of type ITag

public static interface **com.bea.compiler.feature.IPropertyInfoFeature.IDescription**

This interface provides information about a node.

Methods

**public java.lang.Object getProperty(
        String name)**
Returns the property identified by name

public java.lang.String getDescription()
Returns the human-readable description

public static interface **com.bea.compiler.feature.IPropertyInfoFeature.ITag**

This interface provides information about a tag

Methods

public java.lang.String getNamespace()
Returns the namespace of the tag

public java.lang.String getName()
Returns the name of the tag

public int getMinOccurs()
Returns the minimum number of times this tag can appear

public int getMaxOccurs()
Returns the maximum number of time this tag can appear, with -1 indicating 'unbounded'.

public java.lang.String getDescription()
Returns the human-readable description

**public java.lang.Object getProperty(
        String name)**
Returns additional properties identified by name

public java.util.Iterator getAttributes()
Returns the list of attributes associated with this tag The iterator elements must be of type IAttribute

public static interface **com.bea.compiler.feature.IPropertyInfoFeature.IAttribute**

This interface provides information about an attribute.

Methods

public java.lang.String getName()
Returns the name of this attribute

public java.lang.Object getDefaultValue()
Returns the default value of this attribute

public boolean isRequired()
Returns true if this attribute is required

public java.lang.String getDescription()
Returns the human-readable description

public java.lang.Object getProperty(
String name)
Returns additional properties identified by name

public com.bea.compiler.feature.IPropertyInfoFeature.IType getType()
Returns the type of that attribute

public static interface **com.bea.compiler.feature.IPropertyInfoFeature.IType**

This interface provides information about a type.

Methods

public int getTypeCode()
Returns the type code

public int getMaxLength()
Returns the maximum allowable length. Must be implemented if this type is a TYPE_TEXT or a TYPE_TEXTLONG .

public java.lang.Comparable getMax()
Returns the maximum value as defined by the specific ordering rules for the type. Must be implemented if this type is a TYPE_DECIMAL , TYPE_INTEGER or TYPE_DATE .

public java.lang.Comparable getMin()
Returns the minimum value as defined by the specific oredering rules for the type. Must be implemented if this type is a TYPE_DECIMAL , TYPE_INTEGER or TYPE_DATE .

public java.util.Iterator getEnumerationElements()
Returns the elements of an enumeration. Must be implemented if this type is a TYPE_ENUMERATION .

public int getDecimalPlaces()
Returns the number of decimal places. Must be implemented if this type is a TYPE_DECIMAL .

public com.bea.compiler.feature.IPropertyInfoFeature.ICustomType getcustomType()
Returns the custom type description. Must be implemented if this type is a TYPE_CUSTOM .

Fields

public static final TYPE_UNKNOWN

public static final TYPE_TEXT

public static final TYPE_TEXTLONG

public static final TYPE_ENUMERATION

public static final TYPE_BOOLEAN

public static final TYPE_DECIMAL

public static final TYPE_INTEGER

public static final TYPE_DATE

public static final TYPE_CUSTOM

public static interface **com.bea.compiler.feature.IPropertyInfoFeature.ICustomType**

This interface provides info about a custom type

Methods

public java.lang.String getName()
Returns the name of the custom type.

public java.lang.Object getTypeValidator()
Returns the custom validator.

public java.lang.Object getTypeBuilder()
Returns the custom builder.

public java.lang.Object getTypeRenderer()
Returns the custom renderer.

public static interface **com.bea.compiler.feature.IPropertyInfoFeature.ITypeFactory**

Factory interface that supports creating different types

Methods

public com.bea.compiler.feature.IPropertyInfoFeature.IType createUnknownType()
Creates a TYPE_UNKNOWN type.

**public com.bea.compiler.feature.IPropertyInfoFeature.IType createTextType(
int maxLength)**
Creates a TYPE_TEXT type.

**public com.bea.compiler.feature.IPropertyInfoFeature.IType createTextLongType(
int maxLength)**
Creates a TYPE_TEXTLONG type.

**public com.bea.compiler.feature.IPropertyInfoFeature.IType createEnumerationType(
Collection values)**
Creates a TYPE_ENUMERATION type. This methods holds onto the array provided.

public com.bea.compiler.feature.IPropertyInfoFeature.IType createBooleanType()
Creates a TYPE_BOOLEAN type.

public com.bea.compiler.feature.IPropertyInfoFeature.IType createIntegerType(
Comparable min,
Comparable max)
Creates a TYPE_INTEGER type.

public com.bea.compiler.feature.IPropertyInfoFeature.IType createDecimalType(
Comparable min,
Comparable max,
int decimalPlaces)
Creates a TYPE_DECIMAL type.

public com.bea.compiler.feature.IPropertyInfoFeature.IType createDateType(
Comparable min,
Comparable max)
Creates a TYPE_DATE type.

public com.bea.compiler.feature.IPropertyInfoFeature.IType createCustomType(
String name,
Object typeValidator,
Object typeBuilder,
Object typeRenderer)
Creates a TYPE_CUSTOM type.

public static class **com.bea.compiler.feature.IPropertyInfoFeature.TypeFactory**

Contains reference to an ITypeFactory singleton object.

Constructors **public IPropertyInfoFeature.TypeFactory()**

Methods **public static com.bea.compiler.feature.IPropertyInfoFeature.ITypeFactory get()**
Returns the instance of an ITypeFactory

public static void set(
IPropertyInfoFeature.ITypeFactory factory)
Sets the instance of the ITypeFactory . This should be called only by implementors of the interface.

public interface **com.bea.compiler.feature.INodeInfoFeature** implements com.bea.compiler.feature.IFeature

Provides information about the language structure of a file.

Methods **public com.bea.compiler.INode getRootNode()**
Returns the root node in the language tree.

public java.util.List getNodes(
int offset)
Returns the language stack at the given offset.

public interface **com.bea.compiler.feature.IFeature**

The interface all features provide to the compiler clients.

Package com.bea.compiler.fileinfo

This package defines how the client accesses information about the project's files and directories. Its classes and interfaces are described below.

public interface **com.bea.compiler.fileinfo.IFileNamespace**

Maintains the namespace of files for a project. This namespace allows compilers to query for files with given names and to look at any exposed content. In both cases, dependencies are added on that information.

Methods

public com.bea.compiler.fileinfo.IFileInfo getRootDirectory()
Returns the root directory.

public com.bea.compiler.fileinfo.IFileInfo findFile(
    String path)
Returns the node for the file with the given path. If the path begins with a '/', then this path is relative to the project. Otherwise, it is relative to the current file.

PHASE: any

public com.bea.compiler.fileinfo.IFileInfo findFile(
    IFileInfo baseFile,
    String path)
Like findFile above, but if a relative path is used, the file will be interpretted relative to the given base file rather than the current file.

PHASE: any

public com.bea.compiler.fileinfo.IFileInfo findFile(
    String path,
    IFileStore.IRoot root)
Returns the node for the file with the given path inside the given root. If the path begins with a '/', then this path is relative to the project. Otherwise, it is relative to the current file.

PHASE: any

public com.bea.compiler.fileinfo.IFileInfo findFile(
    IFileInfo baseFile,
    String path,
    IFileStore.IRoot root)
Like findFile above, but if a relative path is used, the file will be interpretted relative to the given base file rather than the current file.

PHASE: any

public com.bea.compiler.fileinfo.IFileInfo getFileInfo(
    IFile file)
Returns the node for the given file.

Fields

public static final NAME
Records the name of this namespace.

public static interface **com.bea.compiler.fileinfo.IFileNamespace.IInfoChange** implements com.bea.compiler.INamespace.IInfoChange

Describes a change that occurred in the type namespace. This includes both the outer type and the entity because a new member can show up in types that inherit it as well as its outer type.

Methods

public com.bea.compiler.fileinfo.IFileInfo getNode()
Returns the node in the tree that changed.

public interface **com.bea.compiler.fileinfo.IFileInfo** implements java.lang.Comparable

Represents a node in the file namespace, which can be either a file or a directory. Each file can provide access to its content in various forms. Each content type is encoded using a MIME-like name.

Methods

public java.lang.String getName()
Returns the name of this file or directory. The root directory will have an empty name.

public java.lang.String getPath()
Returns the path to this file or directory, with '/' as separator.

public com.bea.compiler.fileinfo.IFileInfo getParent()
Returns the parent of this file in the tree (or null if this is the root directory).

public boolean isFile()
Determines whether this is a file.

public boolean isDirectory()
Determines whether this is a directory.

public long getLastModified()
Returns the last modified time of a file or -1 for a directory.

public java.util.Iterator getChildren()
Returns the children of this directory.

public java.util.Iterator getChildren(
String name)
Returns the children with the given name. There may be more than one in the various source and and binary paths. However, they will always be returned in their order on the paths (with source before binary).

public com.bea.compiler.fileinfo.IFileInfo getChild(
String name)
Returns the child with the given name or null if none exists. If more than one exists, this will return first one in the list.

public com.bea.compiler.IFile getFile()
Returns the file that this node represents.

public boolean hasContentOfType(
String type)
Determines whether this file has content of the given type.

public java.lang.String[] getContentTypes()
Returns a list of the supported content types for this file.

public java.lang.Object getContent(
String type)
Returns the content of this file in the given type or null if the type is not supported.

Package com.bea.compiler.info

This package defines how the client accesses information about the structure of a source file. Its classes and interfaces are described below.

public class **com.bea.compiler.info.Type**

Contains references to the static types defined. These statics are filled in at run-time by the framework.

Constructors	**public Type()**
Fields	**public static ERROR**
	public static VOID
	public static NULL
	public static BOOLEAN
	public static BYTE
	public static SHORT
	public static INT
	public static LONG
	public static CHAR
	public static FLOAT
	public static DOUBLE
	public static EMPTY_TUPLE

public class **com.bea.compiler.info.Name**

Contains references to the static names defined. These statics are filled in at run-time the framework.

Constructors	**public Name()**
Fields	**public static LANG**
	public static OBJECT

public static CLASS

public static THROWABLE

public static EXCEPTION

public static RUNTIME_EXCEPTION

public static ERROR

public static CLONEABLE

public static STRING

public static SERIALIZABLE

public static XSCRIPT_VARIANT

public static XSCRIPT_ARRAY

public static XSCRIPT_BOOLEAN

public static XSCRIPT_DATE

public static XSCRIPT_FUNCTION

public static XSCRIPT_MATH

public static XSCRIPT_NUMBER

public static XSCRIPT_OBJECT

public static XSCRIPT_REGEXP

public static XSCRIPT_STRING

public static XSCRIPT_XML

public static XSCRIPT_XMLLIST

public interface **com.bea.compiler.info.IVariable**

Encapsulates information about a variable defined in the source. This variable can be a local variable, a method parameter, a field of a user type, and so on.

Methods	**public java.lang.String getID()** Returns the name of this variable.
	public com.bea.compiler.info.IType getType() Returns the type of this variable.
	public boolean isField() Determines whether this variable is a field of a class. If so, this can be safely cast into an IField to get more information.
	public boolean isFinal() Determines whether this variable can be reassigned.

public interface **com.bea.compiler.info.IType**

Represents a Type available in the Java runtime or the error type, which is used at compile time to denote something of an unknown type.

Methods

public boolean isErrorType()
Determines whether this denotes the error type.

public boolean isVoidType()
Determines whether this denotes the void type.

public boolean isNullType()
Determines whether this is the null type.

public boolean isPrimitiveType()
Determines whether this is a primitive type.

public boolean isPrimitiveNumericType()
Determines whether this is a numeric primitive type.

public boolean isPrimitiveIntegralType()
Determines whether this is an integral primitive type.

public int getTypeCode()
Returns the code for a primitive type.

public boolean isReferenceType()
Determines whether this is a reference type.

public boolean isUserType(
IName name)
Determines whether this is the user type with the given name.

public boolean isTupleType()
Determines whether this is a tuple of types.

public int getComponentCount()
Returns the number of components of a tuple type.

public com.bea.compiler.info.IType[] getComponentTypes()
Returns the components of a tuple type.

public boolean isMethodType()
Determines whether this is a method type.

public com.bea.compiler.info.IType getArgumentTypes()
Returns the tuple of argument types of a method type.

public com.bea.compiler.info.IType getReturnType()
Returns the return type of a method type.

public java.lang.String getSignature()
Returns the encoded signature of this type.

public java.lang.String getDisplayName()
Returns the name of this type as it might appear in the source code.

public boolean equals(

Object o)
Determines whether this type is equivalent to the given one.

public int hashCode()
Returns a hash code that is compatible with equals.

Fields	**public static final BOOLEAN** Constant for the boolean type.
	public static final BYTE Constant for the byte type.
	public static final SHORT Constant for the short type.
	public static final INT Constant for the int type.
	public static final LONG Constant for the long type.
	public static final CHAR Constant for the char type.
	public static final FLOAT Constant for the float type.
	public static final DOUBLE Constant for the double type.

public interface **com.bea.compiler.info.IReferenceTypeMember** implements com.bea.compiler.info.IDeclaredEntity

Represents a member of a type, which can be a field, method, or a type.

Methods	**public java.lang.String getID()** Returns the ID of this member.
	public boolean isAmbiguous() Determines whether multiple members with this ID or signature exist.
	public int getModifiers() Returns all of the modifiers in an bitfield (bits defined below).
Fields	**public static final STATIC**
	public static final ABSTRACT
	public static final FINAL
	public static final NATIVE
	public static final SYNCHRONIZED
	public static final TRANSIENT
	public static final VOLATILE

public static final STRICTFP

public interface **com.bea.compiler.info.IReferenceType** implements com.bea.compiler.info.IPackageMember, com.bea.compiler.info.IReferenceTypeMember, com.bea.compiler.info.IType

Represents a reference type, which can be either user-defined type (i.e., a class or interface) or an arrayp type.

Methods

public com.bea.compiler.IFile getFile()
Returns the file containing the declaration of this type.

public com.bea.compiler.info.IPackage getPackage()
Returns the containing package.

public com.bea.compiler.info.IReferenceType getTopType()
Returns the containing top-level type or null if none exists.

public boolean isClass()
Determines whether this type is a class.

public boolean isInterface()
Determines whether this type is an interface.

public boolean isArray()
Determines whether this type is an array.

public boolean hasCanonicalName()
Determines whether this class has a canonial name. Without one, this type cannot be referenced from outside of its original scope.

public java.lang.String getName(
char packageSeparator,
char typeSeparator)
Returns the name of this class a single string with the given characters placed after package and type IDs, respectively, when separating them from further parts of the name.

public boolean isNested()
Determines whether this type is nested in an outer type.

public boolean isInnerClass()
Determines whether this class has an enclosing instance at runtime.

public com.bea.compiler.info.IReferenceType getSuperclass()
Returns the direct superclass of this type. For an interface, this is always java.lang.Object .

public java.util.Iterator getInterfaces()
Returns the direct super-interfaces of this type.

public boolean isSubclassOf(
IName name)
Determines whether this type is a class of the given class.

public boolean isImplementorOf(
IName name)
Determines whether this type is implements the given interface.

public java.util.Iterator getFields()

Returns the field members of this type, including inheritted ones.

public com.bea.compiler.info.IField getField(
String name)
Returns the field with the given name or null if none exists.

public java.util.Iterator getMethods()
Returns the method members of this type, including inheritted ones.

public java.util.Iterator getMethods(
String name)
Returns the methods of this type with the given name, including inheritted ones.

public java.util.Iterator getMethods(
String name,
int argCount)
Returns the methods of this type with the given name and number of arguments, including inheritted ones.

public com.bea.compiler.info.IMethod getMethod(
String name,
IType argTypes)
Returns the method with the given name and argument types or null if none exists.

public com.bea.compiler.info.IMethod getMethod(
String name)
Returns a method with the given name or null if none exist.

public java.util.Iterator getTypes()
Returns the type members of this type, including inheritted ones.

public com.bea.compiler.info.IReferenceType getType(
String name)
Returns the member type with the given name or null if none exists.

public java.util.Iterator getConstructors()
Returns the constructors of this type.

public java.util.Iterator getConstructors(
int argCount)
Returns the constructors of this type with the given number of arguments.

public com.bea.compiler.info.IConstructor getConstructor(
IType argTypes)
Returns the constructor with the given signature or null if none exists.

public boolean isAbstract()
Determines whether this method is not implemented.

public boolean isStatic()
Determines whether this type has an outer type at run-time.

public boolean isFinal()
Determines whether this type can be subclassed.

public boolean isStrictFP()
Determines whether this classes uses strict floating point operations.

public boolean isSynthetic()
Determines whether this class is not visible to other classes at compile-time. Such classes do not show up in the package tree but can appear in the type list of a file.

public interface **com.bea.compiler.info.IPackageMember** implements java.lang.Comparable

Represents a member of a package, which can be either a package or a top-level type.

Methods	**public com.bea.compiler.info.IName getName()** Returns the name of this member.
	public boolean isPackage() Determines whether this is a package.
	public boolean isUserType() Determines whether this is a user-defined type.

public interface **com.bea.compiler.info.IPackage** implements com.bea.compiler.info.IPackageMember

Represents a package that exists in the source code.

Methods	**public com.bea.compiler.info.IPackage getParentPackage()** Returns the containing package or null if none exists.
	public java.util.Iterator getMembers() Returns the members of this package.
	public java.util.Iterator getCurrentMembers() Returns the members of this package that have been found so far.
	public com.bea.compiler.info.IPackageMember getMember(String name) Returns the member with the given name.

public interface **com.bea.compiler.info.IName** implements java.lang.Comparable

Refers to a declared entity in the source code. A name is composed of sequence of identifiers. The last identifier, which is most specific to this named entity, is referred to simply as its ID.

Methods	**public java.lang.String getID()** Returns the most-specific identifier in this name.
	public com.bea.compiler.info.IName getQualifier() Returns the qualifier on this name or null if none exists.
	public java.lang.String[] getParts() Returns all of the identifiers in this name.
	public java.lang.String toString() Returns all of the identifiers with periods concatenating them.

public boolean equals(
Object o)
Determines whether this name is the same as the one given.

public int hashCode()
Returns a hash code compatible with the equals method.

public interface **com.bea.compiler.info.IMethod** implements com.bea.compiler.info.IReferenceTypeMember, com.bea.compiler.info.ICallable

Represents a method of a user-defined type.

Methods

public com.bea.compiler.info.IType getReturnType()
Returns the return type of this method.

public java.util.Iterator getExceptionTypes()
Returns an iterator over the exception types.

public boolean isAbstract()
Determines whether this method is not implemented.

public boolean isStatic()
Determines whether this is a class method.

public boolean isFinal()
Determines whether this method can be overriden.

public boolean isSynchronized()
Determines whether this method is automatically synchronized.

public boolean isNative()
Determines whether this method is implemented in native code.

public boolean isStrictFP()
Determines whether this method uses strict floating point operations.

public interface **com.bea.compiler.info.IField** implements com.bea.compiler.info.IReferenceTypeMember, com.bea.compiler.info.IVariable, java.lang.Comparable

Represents a field of a user-defined type.

Methods

public com.bea.compiler.info.IType getType()
Returns the type of this field.

public boolean isStatic()
Determines whether this is a class field.

public boolean isFinal()
Determines whether this field can be modified.

public boolean isTransient()
Determines whether this field is serialized.

public boolean isVolatile()
Determines whether this field can change value unexpectedly.

public interface **com.bea.compiler.info.IDeclaredEntity**

Represents any entity declared in the source file. This can be a user-defined type, a field, a method, or a constructor. Note that only the first three of those are members of the outer type.

Methods

public com.bea.compiler.info.IReferenceType getOuterType()
Returns the type of which this entity is a member.

public com.bea.compiler.info.IAccess getAccess()
Returns the accessability of this entity.

**public boolean isAccessibleFrom(
IPackageMember location)**
Determines whether this entity is accessible from the given location.

**public boolean isAccessibleFrom(
IPackageMember location,
IAccess access)**
Determines whether this entity would be accessible from the given location if it's access were as given.

public boolean isUserType()
Determines whether this entity is a type.

public boolean isField()
Determines whether this entity is a field.

public boolean isMethod()
Determines whether this entity is a method.

public boolean isConstructor()
Determines whether this entity is a constructor.

public com.bea.compiler.info.IAnnotationSet getAnnotations()
Returns the annotations on this entity.

public interface **com.bea.compiler.info.IConstructor** implements com.bea.compiler.info.ICallable

Represents a constructor of a user-defined class.

public interface **com.bea.compiler.info.ICallable** implements com.bea.compiler.info.IDeclaredEntity, java.lang.Comparable

Represents a callable object in a class, namely a constructor of method.

Methods

public com.bea.compiler.info.IType getArgumentTypes()

Returns the types of the arguments of this method.

public java.util.Iterator getExceptionTypes()
Returns the types that can be thrown by this method.

public java.util.Iterator getArguments()
Returns the arguments of this method.

public int getArgumentCount()
Returns the number of arguments this method has.

public interface **com.bea.compiler.info.IArrayType** implements com.bea.compiler.info.IReferenceType

Represents an array reference type.

Methods

public com.bea.compiler.info.IType getElementType()
Returns the element type of the array.

public interface **com.bea.compiler.info.IArgument** implements java.lang.Comparable

Represents an argument to a callable in the source.

Methods

public java.lang.String getName()
Returns the name of this argument.

public com.bea.compiler.info.IType getType()
Returns the type of this argument.

public interface **com.bea.compiler.info.IAnnotationSet**

Represents the set of annotations and javadoc information on a declared entity, including "synthetic" annotations that were added but not present in the source text proper.

Methods

public java.lang.String getShortDescription()
Returns a short (one sentence) description of this entity.

public java.lang.String getDescription()
Returns a full description of this entity.

public java.util.Iterator iterator()
Returns all of the annotations on this entity.

public com.bea.compiler.info.IAnnotation get(
String[] path)
Returns the annotation with the given path or null if none exists. If more than one exists, an assertion will fire. (The grammar for the annotations will ensure that only one is produced.)

public java.util.Iterator getAll(

String[] path)
Returns all of the annotations with the given path.

public interface **com.bea.compiler.info.IAnnotation**

Contains additional information provided in the language about a particular reference type or member.

Annotations have values that are arbitrary objects. A grammar for a set of annotations will define more precisely what these values will be so that consumers of those annotations can deal with them appropriately.

Methods

public com.bea.compiler.info.IAccess getAccess()
Returns the accessability range of this annotation.

public java.lang.String getID()
Returns the ID of the annotation.

public java.lang.String[] getPath()
Returns the path of names to this annotation from the root.

public java.lang.Object getValue()
Returns the value of the annotation.

Fields

public static final ROOT_NAME
Defines the name of the root node in the annotation tree. This contains the ' and ' characters which are illegal in normal tag names.

public interface **com.bea.compiler.info.IAccess**

Represents a type of accessibility available in the Java runtime on a class, interface, or member.

Methods

public boolean isPublic()
Determines whether this denotes public access.

public boolean isProtected()
Determines whether this denotes protected access.

public boolean isPackage()
Determines whether this denotes package (default) access.

public boolean isPrivate()
Determines whether this denotes private access.

public boolean isAccessibleFrom(
IPackageMember entityLoc,
IPackageMember queryLoc)
Determines whether a member with this access in the given package or class is accessible from the given location.

public boolean isMoreRestrictive(
IAccess access)
Determines whether this access is more restrictive than the given one.

public boolean isLessRestrictive(
IAccess access)
Determines whether this access is less restrictive than the given one.

	public java.lang.String toString() Returns the name of this access.
	public int getID() Returns the ID of this type of access. This is handy for switches.
Fields	**public static final ID_PUBLIC**
	public static final ID_PROTECTED
	public static final ID_PACKAGE
	public static final ID_PRIVATE

public class **com.bea.compiler.info.Access**

Contains references to the four types of access defined. These statics are filled in at run-time the framework.

Constructors	**public Access()**
Fields	**public static PUBLIC**
	public static PROTECTED
	public static PACKAGE
	public static PRIVATE

Package com.bea.compiler.info.beaninfo

This package defines how the client accesses additional information available for JavaBeans. Its classes and interfaces are described below.

public interface **com.bea.compiler.info.beaninfo.IPropertyDescriptor**

Represents a JavaBean-style property.

Methods	**public java.lang.String getName()** Returns the name of this property
	public com.bea.compiler.info.IType getPropertyType() Returns the type of this property
	public com.bea.compiler.info.IMethod getReadMethod() Returns the read method for this property, null if the property is write-only
	public com.bea.compiler.info.IMethod getWriteMethod() Returns the write method for this property, null is the property is read-only

public class **com.bea.compiler.info.beaninfo.Introspector**

Constructors	**public Introspector()**
Methods	**public static void set(** **IIntrospector i)** Sets the driver for the IBeanInfo class; this is to be called only by the framework **public static com.bea.compiler.info.beaninfo.IBeanInfo getBeanInfo(** **IReferenceType type)** Returns the IBeanInfo for this type

public interface **com.bea.compiler.info.beaninfo.IIntrospector**

Methods	**public com.bea.compiler.info.beaninfo.IBeanInfo getBeanInfo(** **IReferenceType type)** Returns the BeanInfo associated to the given type. This can be then interrogated for properties etc.

public interface **com.bea.compiler.info.beaninfo.IIndexedPropertyDescriptor** implements com.bea.compiler.info.beaninfo.IPropertyDescriptor

Represents a JavaBean-style indexed property.

Methods	**public com.bea.compiler.info.IType getIndexedPropertyType()** Returns the type of the indexed property **public com.bea.compiler.info.IMethod getIndexedReadMethod()** Returns the read method of the indexed property **public com.bea.compiler.info.IMethod getIndexedWriteMethod()** Returns the write method of the indexed property

public interface **com.bea.compiler.info.beaninfo.IEventDescriptor**

Represents a JavaBean-style event descriptor

Methods	**public com.bea.compiler.info.IReferenceType getListenerType()** Returns the target interface that will get invoked when the event is fired **public com.bea.compiler.info.IMethod getAddListenerMethod()** Returns the method used to register a listener at the event source **public com.bea.compiler.info.IMethod getRemoveListenerMethod()** Returns the method used to unregister a listener from the event source **public boolean isUnicast()**

Returns true if the event source is unicast, false if it's multicast

public interface **com.bea.compiler.info.beaninfo.IBeanInfo**

Represents the JavaBean information associated with a specific reference type

Methods	**public com.bea.compiler.info.beaninfo.IPropertyDescriptor[] getPropertyDescriptors()** Returns descriptors for the JavaBean properties exposed by this BeanInfo
	public com.bea.compiler.info.beaninfo.IEventDescriptor[] getEventDescriptors() Returns descriptor for the JavaBean-style events fired by this bean

Package com.bea.compiler.java

This package defines the information produced by the Java programming language support in the compiler. Its classes and interfaces are described below.

public class **com.bea.compiler.java.JavaStructure** extends com.bea.compiler.feature.IStructureFeature.StructureCode

Methods	**public static com.bea.compiler.feature.IStructureFeature.StructureCode getCode(int code)**
Fields	**public static final UNIT_CODE**
	public static final PACKAGE_CODE
	public static final IMPORT_CODE
	public static final CLASS_CODE
	public static final FIELD_CODE
	public static final METHOD_CODE
	public static final CONSTRUCTOR_CODE
	public static final PARAM_CODE
	public static final NAME_CODE
	public static final TYPE_CODE
	public static final ACCESS_CODE
	public static final STATIC_CODE
	public static final ABSTRACT_CODE
	public static final FINAL_CODE

public static final NATIVE_CODE

public static final SYNCHRONIZED_CODE

public static final TRANSIENT_CODE

public static final VOLATILE_CODE

public static final STRICTFP_CODE

public static final ONDEMAND_CODE

public static final ISINTERFACE_CODE

public static final SUPERTYPE_CODE

public static final INTERTYPES_CODE

public static final THROWS_CODE

public static final BODY_CODE

public static final UNIT
Represents a compilation unit.

Children: PACKAGE, IMPORT, CLASS

Properties: NONE

public static final PACKAGE
Represents a package node.

Children: NONE

Properties: NAME

public static final IMPORT
Represents an import node.

Children: NONE

Properties: NAME, ONDEMAND

public static final CLASS
Represents a class node (top-level or inner).

Children: FIELD, METHOD, CLASS

Properties: NAME, all modifiers, ACCESS

public static final FIELD
Represents a field node.

Children: NONE

Properties: NAME, TYPE, all modifiers, ACCESS

public static final METHOD
Represents a method node.

Children: PARAM

Properties: NAME, TYPE, all modifiers, ACCESS

public static final CONSTRUCTOR
Represents a constructor node.

Children: PARAM

Properties: THROWS, ACCESS

public static final PARAM
Represents a formal parameter node.

Children: NONE

Properties: NAME, TYPE, all modifiers

public static final NAME
The name property (String).

public static final TYPE
The type property (String).

public static final ACCESS
The acessibility property (String).

public static final STATIC
The static modifier property (Boolean).

public static final ABSTRACT
The abstract modifier property (Boolean).

public static final FINAL
The final modifier property (Boolean).

public static final NATIVE
The native modifier property (Boolean).

public static final SYNCHRONIZED
The synchronized modifier property (Boolean).

public static final TRANSIENT
The transient modifier property (Boolean).

public static final VOLATILE
The volatile modifier property (Boolean).

public static final STRICTFP
The strictfp modifier property (Boolean).

public static final ONDEMAND
The import-on-demand property (Boolean).

public static final ISINTERFACE
The isInterface property for a class (Boolean).

public static final SUPERTYPE
The super class property for a class. (String).

public static final INTERTYPES
The super interfaces property for a class. (String[])

public static final THROWS

The exceptions property for a method. (String[])

public static final BODY
The body property for a method. (String)

Package com.bea.compiler.java.annotation

This package defines the information produced by the Java programming language support in the compiler about the annotations found in Java source code. Its classes and interfaces are described below.

public class **com.bea.compiler.java.annotation.AnnotationStructure** extends com.bea.compiler.feature.IStructureFeature.StructureCode

Methods	**public static com.bea.compiler.feature.IStructureFeature.StructureCode getCode(int code)**
Fields	**public static final COMMENT_CODE**
	public static final TAG_CODE
	public static final ATTRIBUTE_CODE
	public static final NAME_CODE
	public static final VALUE_CODE
	public static final COMMENT
	public static final TAG
	public static final ATTRIBUTE
	public static final NAME
	public static final VALUE

Package com.bea.compiler.typeinfo

This package defines how the client accesses information about types. Its classes and interfaces are described below.

public interface **com.bea.compiler.typeinfo.ITypeNamespace** implements com.bea.compiler.INamespace

Represents the namespace of types that were defined in the project.

Methods	**public com.bea.compiler.info.IPackage getPackageRoot()**

Returns the root package object.

public com.bea.compiler.info.IName getName(
String name)
Returns a name object for the given qualified name.

public com.bea.compiler.info.IPackageMember getEntity(
IName name)
Returns the package or type with the given name.

public com.bea.compiler.info.IPackage getPackage(
IName name)
Returns the package with the given name. This will return null if the name is that of a type.

public com.bea.compiler.info.IReferenceType getUserType(
IName name)
Returns the class or interface with the given name. This will return null if the name is that of a package.

public java.util.Iterator getTypes(
IFile file)
Returns the types of the given file.

Fields

public static final NAME
Records the name of this namespace.

public static interface **com.bea.compiler.typeinfo.ITypeNamespace.IInfoChange** implements com.bea.compiler.INamespace.IInfoChange

Describes a change that occurred in the type namespace. This includes both the outer type and the entity because a new member can show up in types that inherit it as well as its outer type.

Methods

public com.bea.compiler.info.IReferenceType getOuterType()
Returns the outer type containing the entity (possibly null).

public com.bea.compiler.info.IDeclaredEntity getEntity()
Returns the entity that changed.

Compiler Framework Language API

This API defines how to extend the compiler with new languages.

Packages	
com.bea.language	Defines how support for compiling a particular language is provided to the compiler framework.
com.bea.language.codegen	Contains the interfaces for the intermediate representation.
com.bea.language.enterprise	Provides information about the enterprise context of the

	project
com.bea.language.feature	Defines how a language extension accesses information about certain features of the source code
com.bea.language.fileinfo	Interfaces implemented by compilers that wish to access or manipulate file information
com.bea.language.info	Defines mutable versions of the compiler info interfaces of `com.bea.compiler.info`.
com.bea.language.java	Defines how the Java language implementation can be extended to create support for other languages based on Java and its annotations.
com.bea.language.schema	Provides interfaces for building parsers and validity checkers for tree shaped data.
com.bea.language.transform	Provides interfaces for building transforming compilers that transform a source language to a target language
com.bea.language.typeinfo	Provides interface for languages that need to access or manipulate type information.
com.bea.language.xscript	Provides interfaces for accessing and manipulating XScript code (JavaScript with native XML support)

This API defines how to extend the compiler with new languages. The package `com.bea.language` contains the main interfaces. Some subpackages of this provide additional interfaces that are specific to a particular language. For example, the APIs in `com.bea.compiler.java` define how to extend Java with further language information.

Package com.bea.language

This exemplary package defines how the compiler framework interacts with specific language modules. It defines how support for compiling a particular language is provided to the compiler framework. Its classes and interfaces are described below.

public interface **com.bea.language.IParseNode**

Implemented by all parse tree nodes in the compiler framework.

Methods	**public IToken getFirstToken()** Returns the first token for this node.
	public IToken getLastToken() Returns the last token for this node.

public interface **com.bea.language.ILineColumn**

Records the line and column of a particular offset in the source.

Methods	**public int getLine()** Returns the one-indexed line number of the offset. **public int getColumn()** Returns the one-indexed column number of the offset.

public interface **com.bea.language.ILangToken**

Extends IToken with the ability to retrieve the text value of the token. This is in the language API because this method is only available during compilation.

The purpose of having this type is to allow for tokens that retrieve their text values in different ways. Thus, this method should be always be used to retrieve the text of a token, rather than asking the ILangContext for the text at this range.

Methods	**public java.lang.String getText()** Returns the text of this token.

public interface **com.bea.language.ILangResolver**

Encapsulates the ability to translate from the parse tree into a set of Java class files.

Methods	**public com.bea.language.ILangResolver.IFileResolver getFileResolver()** Returns a resolver for the entire file. **public com.bea.language.ILangResolver.IUpdateResolver getUpdateResolver()** Returns an update resolver for this file.

public static interface **com.bea.language.ILangResolver.IFileResolver**

Performs resolving of the file as a whole.

Methods	**public com.bea.language.ILangResolver.ITypeResolver getTypeResolver(IReferenceType topType)** Returns a resolver for a top-level type.

public static interface **com.bea.language.ILangResolver.ITypeResolver**

Performs resolving of a type in the file.

Methods	**public com.bea.language.ILangResolver.ITypeResolver getTypeResolver(IReferenceType innerType)**

Returns a resolver for a nested type.

public com.bea.language.info.ILangUserType getType()
Returns the type being resolved.

public void resolveBegin()
Called before resolving commences on this type.

public void resolveModifiers()
Fills in modifiers of the type.

public com.bea.language.ILangResolver.ITypeResolver.SuperTypes resolveSuperTypes()
Returns the names of the super types of the given type.

public com.bea.language.info.ILangAnnotationSet resolveAnnotations()
Resolves the annotations on this type. This is done after resolving super types so that they may be used in resolving annotations.

public java.util.Map resolveMemberTypes()
Returns the types declared in the given type indexed by name.

public java.util.Map resolveMemberFields()
Returns the fields declared in the given type indexed by name.

public java.util.Map resolveMemberMethods()
Returns the methods declared in the given type indexed by signature.

public java.util.Map resolveConstructors()
Returns the constructors declared in the given type indexed by argument types.

**public com.bea.language.info.IConstant resolveConstant(
ILangField field)**
Returns the constant value for the given field or null if it does not have a constant value.

public void resolveEnd()
Called when resolving finishes on this type.

public static class **com.bea.language.ILangResolver.ITypeResolver.SuperTypes**

Records the superclass and interfaces of some type.

Constructors	**public ILangResolver.ITypeResolver.SuperTypes(IReferenceType superclass, List interfaces)**
Fields	**public superclass**
	public interfaces

public static interface **com.bea.language.ILangResolver.IUpdateResolver**

Used to update links between entities and their source in the parse tree when copying resolve shapes.

Methods

public void updateType(
    ILangUserType type)
Updates any links to this type from it's source.

public void updateField(
    ILangField field)
Updates any links to this field from it's source.

public void updateMethod(
    ILangMethod method)
Updates any links to this method from it's source.

public void updateConstructor(
    ILangConstructor constructor)
Updates any links to this constructor from it's source.

public static interface **com.bea.language.ILangResolver.IOuterContext**

Contains information passed by the outer language during parsing.

Methods

public com.bea.language.info.IScope getScope()
Returns the scope enclosing the nested language.

public static class **com.bea.language.ILangResolver.DefaultOuterContext** implements com.bea.language.ILangResolver.IOuterContext

Provides a default implementation of IOuterContext .

Constructors

public ILangResolver.DefaultOuterContext(
    IScope scope)

Methods

public com.bea.language.info.IScope getScope()

public static class **com.bea.language.ILangResolver.OuterContext**

Provides a static reference to the current outer resolved.

Constructors

public ILangResolver.OuterContext()

Methods

public static com.bea.language.ILangResolver.IOuterContext get()
Returns the outer context for the language being resolved.

public static void set(
    ILangResolver.IOuterContext c)
Stores the outer context for the language being resolved.

public abstract class **com.bea.language.ILangReader** extends java.io.Reader

Extends the Reader class to allow a transducer to be used as input to a scanner. The transducer may hide multiple characters in the real stream with a single character (or vice versa), so this interface provides a way for the transducer to communicate the actual size in the underlying stream of the characters returned.

Constructors **public ILangReader()**

Methods **public int read(**
char[] cbuf,
int offset,
int len)
With the added information of this class, this method is no longer adequate if more than one character is requested because there is no way to return intermediate size changes.

This implementation is provided to allow single-character reads via this interface. However, new code should be written to call the new read method below.

public int getSize()
Returns the real size in the underlying stream of character just returned via one of the normal read methods of Reader .

public abstract int read(
char[] cbuf,
int[] sbuf,
int offset,
int len)
Extends the main read method of Reader with information about the size of the characters returned in the underlying stream.

public interface **com.bea.language.ILangParser**

Encapsulates the parsing algorithm for a language.

Methods **public void parse()**
Performs a parse of the file.

public static interface **com.bea.language.ILangParser.IOuterContext**

Contains information passed by the outer language during parsing.

public static class **com.bea.language.ILangParser.OuterContext**

Provides a static reference to the current outer context.

Constructors **public ILangParser.OuterContext()**

Methods **public static com.bea.language.ILangParser.IOuterContext get()**
Returns the outer context for the language being parsed.

public static void set(
ILangParser.IOuterContext c)
Stores the outer context for the language being parsed.

public interface **com.bea.language.ILangNode**

Represents a node in the tree of nested languages. Each language owns some range of the file.

Methods

public java.util.Iterator getTokens()
Returns the tokens for this language in forward order.

public java.lang.Object getParseTree()
Returns the parse tree for this language.

public void parse(
ILangParser.IOuterContext context)
Performs parsing in the nested languagge.

public void check(
ILangChecker.IOuterContext context)
Performs checking in the nested language.

public interface **com.bea.language.ILangNamespace**

Defines the required implementation for a namespace of some sort that will be managed as part of a project. Namespaces are fundamental to compilation: the information they record is used by languages in order to ensure correct cooperation between files and is used by the client in order to understand what sort of objects are defined in a source file. As a result, namespaces must interact with the compiler in more detailed ways than even languages.

Each registered namespace will be instantiated using the default constructor in order to create an instance for a new project.

A bit of background about compilation is relevant here. When compilation begins, a list of changed files is known. During compilation, a namespace may discover that a dependency of another file has been broken. This will add an out-of-date file into the compilation. Also during compilation, a namespace may decide to add a file into the project. This normally occurs because a reference was made to a name that does not exist in the project but for which a file is found that probably contains the object with that name. This will add a "discovered" file into the compilation. Thus, at the end of compilation, there may be more files compiled than at the beginning.

Namespaces are responsible for updating themselves to the results of compilation of individual files and for notifying the compiler of out-of-date and discovered files.

File Status

Each time a compilation is performed, there is a target file status that is needed for each file. A "one-hot" encoding is used to indicate the status of a file. However, the status variable passed this instance will actually encode a range of statuses, meaning that compilation will stop when the file reaches any one of them.

Methods

public void setContext(

ILangNamespace.IContext context)
Gives the namespace the context through which it can access namespace-related APIs from the project.

public java.lang.Object getInitialRecord(
IFile file)
Returns an initial record for the given file about to be added. Note that the namespace must not update itself to this file until it is actually compiled.

public void startCompile(
Collection files,
int status)
Allows the namespace to prepare for compilation of the given files. Additional files may be added or discovered to be out of date during the compile.

public void endCompile(
Collection files,
int status)
Allows the namespace to clean up from compilation of the given files. This list will match the list passed to beginCompile . The complete list of compiled files can be retrieved from the context.

public void startCompile(
IFile file)
Allows the namespace to prepare for compilation of the given file, which is just about begin compilation. This may be a changed, discovered, or out-of-date file.

public void endCompile(
IFile file)
Allows the namespace to clean up from compilation of the given file.

public void cleanRecord(
IFile file,
Object record)
Allows the namespace to clean out compile-time-only information from a record on the given file.
Parameters
file - The file whose record is being cleaned.
record - The record on the file that is being cleaned.

public void updateFromParse(
IFile file)
Called just after a file is parsed. The namespace will typically update to the list of objects found to be defined in the file.

public void resolve(
IFile file)
Perform all necessary resolution for this namespace and this file. Note that this will be called for removed files as well because the namespace needs to look for broken dependencies in that case also.

Fields

public static final STATUS_INITIAL
Indicates that nothing is known about the contents of the file.

public static final STATUS_PARSED
Indicates that the names of the objects defined in the file are known.

public static final STATUS_RESOLVED
Indicates that all information about the file's namespace objects are known.

public static final STATUS_CHECKED
Indicates that all errors have been found in the file.

public static final STATUS_GENERATED
Indicates that a .class file has been produced for the file.

public static interface **com.bea.language.ILangNamespace.IInfoChange**

Represents a change that was found in the namespace during the current compile. Changes are recorded in order to determine whether dependencies have been broken as well as to inform the client. Not all recorded changes need to be reported to the client.

Methods

public IFile getFile()
Returns the file in which this change occurred.

public boolean isReported()
Determines whether this change should be reported to the client.

public static interface **com.bea.language.ILangNamespace.IContext**

Defines the additional APIs available to namespaces.

Because namespaces can be used both during compilation and outside of it (i.e., from a feature), some of these methods must be available at both times. Methods not saying otherwise are available at all times.

Methods

public IProject getProject()
Returns the project that owns this namespace.

public long getLastCompiled()
Returns the internal "timestamp" of the most recently started compile of this project.

public long getLastCompiled(
IFile file)
Returns the timestamp for the given file.

public void addChange(
ILangNamespace.IInfoChange change)
Adds the given change in this namespace into the change list. This must only be called during a compilation.

public void parse(
IFile file)
Ensures that the given file has been parsed.

public void resolve(
IFile file,
ILangNamespace.IInfoResolver resolver)
Ensures that the given resolution has been performed. This may be called during compilation or not.

public IFile addDiscoveredFile(
IFileStore.IEntry entry)
Adds the given file into the project. If this is called during compilation, this file will automatically be added to the list of files to be updated. The return value will be null if this file is already in the project.

public void addOutOfDateFile(

IFile file)
Records that the given file has been discovered to be out-of-date and schedules it to be updated in the current compile. This must only be called during a compilation.

public java.util.Collection getCompiledFiles()
Returns all files updated during this compilation. This includes changed, discovered, and out-of-date files.

public java.lang.Object getRecord(
IFile file)
Returns the namespace-specific information recorded for this file in the most recent compile.

public java.lang.Object getPreviousRecord(
IFile file)
Returns the namespace-specific information recorded for this file in the previous compile.

public void setRecord(
IFile file,
Object val)
Sets the namespace-specific information for this file.

public java.util.Iterator getDependencies(
IFile file)
Returns all dependencies of this namespace on the given file.

public void addDependency(
IFile source,
IDependency dep)
Adds the given dependency whose source is the given file.

public void addDiagnostic(
IFile file,
String code,
IToken first,
IToken last,
Object[] args)
Adds a diagnostic to the given file.

public boolean isBroken(
IDependency dep)
Determines whether the given dependency has been broken by changes found so far.

public boolean isBrokenBy(
IDependency dep,
IFile file)
Determines whether the given dependency has been broken by changes found in the given file.

public IFileStore.IEntry findEntry(
String path)
Searches for an entry on the source or binary path with the given name, ignoring extension. If multiple such files exist, the rules for choosing one are the following:
Source files trump binary files.

Files trump directories.

Extensions are tried in priority order.

public java.util.Collection findEntries(
String dir)
Returns all entries under the given directory.

public IFileStore.IEntry getSourceEntry(
String name,
boolean isDir)
Returns the entry for the source file with the given name.

public IFileStore.IEntry getBinaryEntry(
String name,
boolean isDir)
Returns the entry for the binary file with the given name.

public java.lang.String getRelativePath(
IFile file)
Returns the relative path to the given file or null if this file is not in any of the roots of the project.

public com.bea.language.ILangLanguage getRootLanguage(
IFile file)
Returns the root language for the given file.

public com.bea.language.ILangCompiler getRootCompiler(
IFile file)
Returns the root compiler for the given file.

public com.bea.language.ILangNode getRootNode(
IFile file)
Returns the root node of the given file.

public java.lang.Object enterRootContext(
IFile file)
Set the context to the root language of the given file.

public void leaveRootContext(
Object oldContext)
Restores the context to its value at entry.

public int getCompilationPhase()
Returns the current phase of compilation, i.e., the status that this file will reach next.

public com.bea.language.ILangNamespace.IInfoResolver getCurrentResolver()
Returns the resolver that is currently running on this thread.

public static interface **com.bea.language.ILangNamespace.IInfoResolver**

Used to perform some type of resolution on some object in the namespace. Every resolve can be described by a pair (object, phase) where object can be any object in the namespace and phase is an integer describing the type of information needed. It is assumed that two resolve requests are equal iff the objects and phases are equal.

Methods

public java.lang.Object getTarget()
Returns the object to be resolved.

public int getPhase()
Returns the type of resolution to occur.

public boolean isComplete()
Determines whether this resolution is already complete. This must be implemented even if all callers verify that the resolution is not complete before requesting it, as race conditions would

otherwise exist.

public void run()
Performs the given resolution on this object.

public abstract class **com.bea.language.ILangMultiReader** extends com.bea.language.ILangReader

Extends ILangReader with the ability to insert the contents of another file into the stream at the current position.

Constructors **public ILangMultiReader()**

Methods **public abstract void pushFile(**
IFileStore.IEntry file)
Inserts the contents of the given file at the current position.

public abstract java.util.Collection getFiles()
Returns the paths of all the files whose contents are being read.

public abstract com.bea.language.ILangToken addNextToken(
int type,
int size)
Returns a token object with the given dimensions. This token will be in the token set of this node only if the current set of files being read contains only the original file.

public interface **com.bea.language.ILangLocator**

Provides the ability to retrieve a text range for a declared entity produced from a declaration in the file. This range is used for various things such as locating errors produced by the framework. The ranges returned from the methods below should just be the range of the name of this identifier, rather than the range of the entire declaration in the source.

Methods **public IToken getFirst(**
ILangDeclaredEntity entity)
Returns the first token for the range of the given entity.

public IToken getLast(
ILangDeclaredEntity entity)
Returns the last token for the range of the given entity.

public IToken getFirst(
ILangAnnotation annotation)
Returns the first token for the range of the given annotation.

public IToken getLast(
ILangAnnotation annotation)
Returns the last token for the range of the given annotation.

public IToken getFirst(
ILangVariable var)
Returns the first token for the range of the given variable.

public IToken getLast(

ILangVariable var)
Returns the last token for the range of the given variable.

public interface **com.bea.language.ILangLanguage**

Extends the ILanguage interface with information needed by the framework.

Methods

public boolean isNameRestricted()
Determines whether this language requires that each file define only one top-level type and that its name must match the relative path to file within the project. If this is true, then the framework will enforce that each file define one public class that matches the relative path. If it is false, then discovery of these files on the source path will not be supported.

public com.bea.language.ILangCompiler getCompiler()
Returns a compiler for this language. A single instance will be used for each file (or more accurately, language node) of this language.

public com.bea.language.feature.ILangFeature getFeature(
Class key)
Returns the feature identified by the given key.

public interface **com.bea.language.ILangGenerator**

Encapsulates the algorithm for generating code in this language. This is accomplished by implenting a transformation from the parse tree of this file into the intermediate language for each class defined.

Methods

public void generateCode()
Performs code generation for this file by calling the back-end to emit the bytecodes for each class defined (in intermediate form).

public static interface **com.bea.language.ILangGenerator.IOuterContext**

Contains information passed by the outer language during codegen.

public static class **com.bea.language.ILangGenerator.OuterContext**

Provides a static reference to the current outer context.

Constructors

public ILangGenerator.OuterContext()

Methods

public static com.bea.language.ILangGenerator.IOuterContext get()
Returns the outer context for the language being codegenned.

public static void set(
ILangGenerator.IOuterContext c)
Stores the outer context for the language being codegenned.

public interface **com.bea.language.ILangContext**

Provides the context in which the stages of compilation are performed. For convenience, all methods are defined here, although some are not applicable in all phases. This is done to allow phases to share code, which would then not know what phase it was running in. If a method is called in an inappropriate phase, an assertion will fire.

Each method below is marked with which phases can use it.

Methods

public IFile getFile()
Returns the file being compiled.

PHASE: any

public IProject getProject()
Returns the project being compiled.

PHASE: any

public java.net.URI getPath()
Returns the path to the file being compiled.

PHASE: any

public IFileStore.IEntry getEntry()
Returns the entry of the file being compiled.

PHASE: any

public INamespace getNamespace(
String name)
Returns the namespace with the given unique name.

PHASE: any

public IPackage getPackageRoot()
Returns the root package object.

PHASE: any

public IPackage getPackage(
IName name)
Returns the package with the given name.

PHASE: any

public IName getClassName()
Returns the expected class name of the current file or null if this class does not have an expected name.

PHASE: any

public IFileInfo findFile(
String path)
Returns the node for the file with the given path. If the path begins with a '/', then this path is relative to the project. Otherwise, it is relative to the current file.

PHASE: any

public IFileInfo findFile(
IFileInfo baseFile,
String path)
Like findFile above, but if a relative path is used, the file will be interpretted relative to the given base file rather than the current file.

PHASE: any

public IFileInfo findFile(
IFileInfo baseFile,
String path,
IFileStore.IRoot root)
Like findFile above, but if a relative path is used, the file will be interpretted relative to the given base file rather than the current file.

PHASE: any

public void addDiagnostic(
String code,
IToken first,
IToken last,
Object[] args)
Adds a new error to this file.

PHASE: parse, resolve, check
Parameters
code - Provides unique key for this error type.
first - Provides the first token of the range of this diagnostic.
last - Provides the last token of this range of this diagnostic.
args - Provides additional information necessary in formatting user messages for this error. In general, this should include only that which is not clear from the text at which the error was found. This can also be null to indicate no arguments.

public java.lang.Object getProperty(
String name,
Object defValue)
Returns the value of the given property or the given default value if it has not been set. Properties set on the project will override the global values for the compiler.

PHASE: any

public com.bea.language.ILineColumn toLineColumn(
int offset)
Returns the line and column of the given offset.

PHASE: any

public java.io.InputStream getInputStream()
Returns an input stream containing the bits of this file.

PHASE: parse

public com.bea.language.ILangReader getReader()
Returns a reader containing the characters of this file.

PHASE: parse

public com.bea.language.ILangMultiReader getMultiReader()
Returns a reader containing the characters of this file but with the added ability to insert the contents of other files into the stream when necessary.

PHASE: parse

public com.bea.language.ILangNode createNestedLanguage(
    ILangLanguage lang,
    int start,
    int end)
Creates and returns a new node in the nested language tree.

PHASE: parse

public ILanguage getLanguage()
Returns the language for the current language node.

PHASE: any

public void resetTokens()
Remove the token set of the current file.

PHASE: parse

public com.bea.language.ILangToken addNextToken(
    int type,
    int size)
Records the next token found by the scanner. This token may or may not be from the token set of the current file.

PHASE: parse

public java.lang.String getText(
    int start,
    int end)
Returns the text of the given token in the source. This should almost never be called by a language implementation.

PHASE: any

public void setParseTree(
    Object obj)
Stores the root node of the parse tree for this file.

PHASE: parse

public void setTypes(
    List types)
Stores the top-level types declared in this file.

PHASE: parse

public com.bea.language.info.ILangInfoFactory getFactory()
Returns the factory for info structures.

PHASE: parse, resolve, check

public IType getType(
    String signature)
Returns a type (created if necessary) for the given signature.

PHASE: parse, resolve, check

public java.lang.Object getParseTree()
Returns the root node of the parse tree for this file.

PHASE: resolve, check, codegen

public java.util.Iterator getTypes()
Returns the top-level types declared in this file.

PHASE: resolve, check, codegen

public IReferenceType getUserType(
IName name)
Returns the class or interface with the given name.

PHASE: resolve, check, codegen

public boolean setExternalFile(
ILangUserType type,
String name)
For a type that is defined in another file (inner .class files only), this is used to correct the path to that file. This is only allowed for binary files. This returns true iff the file can actually be found.

PHASE: resolve (binary)

public void generateCodes(
IRClassDeclaration decl)
Causes code to be generated for the given class (including nested classes).

PHASE: codegen

public void generateCodes(
IParseNode imports,
IRStatement statements)
Produces binary or source codes for a given set of functions and statements at level 0 used by XScript.

PHASE: codegen

public java.io.Writer getWriterForSourceCodes(
IRClassDeclaration clazz)
Returns a writer to the file where generated source codes for a user type with the given definition will be placed. The framework will record that this file was generated during this compilation.

NOTE: this is for use by back-ends , not language implementations.

PHASE: codegen

public java.io.OutputStream getStreamForBinaryCodes(
IRClassDeclaration clazz)
Returns an output stream to the file where generated binary codes for a user type with the given definition will be placed or null if this class should not be generated. The framework will record that this file was generated during this compilation.

NOTE: this is for use by back-ends , not language implementations.

PHASE: codegen

public interface **com.bea.language.ILangCompiler**

Encapsulates the ability to compile from a source language into Java class files.

Methods	**public com.bea.language.ILangLocator getLocator()** Returns a locator that can provide text ranges for type information produced by compilers of this language.
	public com.bea.language.ILangParser getParser() Returns a parser to use for the first phase of compilation.
	public com.bea.language.ILangResolver getResolver() Returns a resolver to use for the second phase of compilation.
	public com.bea.language.ILangChecker getChecker() Returns a checker for the third phase of compilation.
	public com.bea.language.ILangGenerator getGenerator() Returns a code generator for the fourth phase of compilation.

public interface **com.bea.language.ILangChecker**

Encapsulates a semantic checking algorithm to perform as part of compilation for a language.

Methods	**public void check()** Performs semantic checking on this language.

public static interface **com.bea.language.ILangChecker.IOuterContext**

Contains information passed by the outer language during parsing.

Methods	**public com.bea.language.info.IScope getScope()** Returns the scope enclosing the nested language.

public static class **com.bea.language.ILangChecker.OuterContext**

Provides a static reference to the current outer resolved.

Constructors	**public ILangChecker.OuterContext()**
Methods	**public static com.bea.language.ILangChecker.IOuterContext get()** Returns the outer context for the language being resolved.
	public static void set(**ILangChecker.IOuterContext c)** Stores the outer context for the language being resolved.

public interface **com.bea.language.ILangBackEnd**

Defines the functionality required in order to provide a back-end to the compiler.

Methods

public void generateCodes(
IRClassDeclaration clazz)
Produces binary or source codes for the given class (including nested classes). The implementation asks the framework (through the ILangContext) for the paths where the generated file(s) are placed. These calls also notify the framework of which files were generated (and hence, modified) during this compile.

public void generateCodes(
IParseNode imports,
IRStatement statements)
Produces binary or source codes for a given set of functions and statements used by XScript.

public interface **com.bea.language.IDependency**

Records information about a dependency of a file (or more specifically, an object in the namespace) on the state of some part of the namespace. Namespaces will often keep references to the namespace on the "target" of the dependency, which is the parent of all objects in which changes would break the dependency. This makes it easy to see if a change broke any dependencies. The framework facilitates this by allowing dependencies to set and remove these references at the appropriate time of compilation.

Methods

public java.lang.String getNamespace()
Returns the name of the namespace that created this dependency.

public boolean equals(
Object o)
Determines whether this dependency is a duplicate of another.

public int hashCode()
Helps to determine whether this dependency is a duplicate.

public void addToTarget()
Add references to this dependency in the target if necessary.

public void removeFromTarget()
Remove any references to this dependency in the target.

public IFile getTargetFile()
Returns the file containing the target (if the target exists).

public com.bea.language.IDependency.IBreakChecker getBreakChecker()
Returns an object that can look for changes that break this.

public static interface **com.bea.language.IDependency.IBreakChecker**

Allows checking for changes that break this dependency.

Methods

public boolean isBrokenBy(
ILangNamespace.IInfoChange change)
Determines whether this dependency is broken by this change.

public class **com.bea.language.CyclicDependencyException** extends java.lang.RuntimeException

Thrown when a cyclic dependency is found in the resolve phase. This is an unchecked exception because it can be thrown almost anywhere.

Constructors	**public CyclicDependencyException()**

public class **com.bea.language.Context**

To simplify life for compiler writers, this class contains a static method that will return the current compilation context. This means that languages do not need to pass the context down through each component.

Constructors **public Context()**

Methods **public static com.bea.language.ILangContext get()**
Returns the context in which compilation is now being performed.

public static void set(
ILangContext c)
Stores the context in which compilation is now being performed.

Package com.bea.language.codegen

This package contains the interfaces for the intermediate representation. Its classes and interfaces are described below.

public interface **com.bea.language.codegen.IRXQRLStatement** implements com.bea.language.codegen.IRStatement

Represents a foriegn set of statement(s) in the source code.

Methods **public com.bea.language.codegen.IRExpression getForeignBlock()**
Returns foreign code block.

public interface **com.bea.language.codegen.IRXMLListExpression** implements com.bea.language.codegen.IRExpression

Methods **public com.bea.language.codegen.IRExpression getExpression()**
Returns the expression.

public interface **com.bea.language.codegen.IRXMLExpression** implements

com.bea.language.codegen.IRExpression

Methods **public com.bea.language.codegen.IRExpression getExpression()**
Returns the expression.

public interface **com.bea.language.codegen.IRXMLDotQuery** implements com.bea.language.codegen.IRExpression

Methods **public com.bea.language.codegen.IRExpression getFirst()**

public com.bea.language.codegen.IRExpression getQueryExpression()

public interface **com.bea.language.codegen.IRWithStatement** implements com.bea.language.codegen.IRStatement

Methods **public com.bea.language.codegen.IRExpression getSelector()**
Returns the list of selectors for the with.

public com.bea.language.codegen.IRStatement getStatement()
Returns the statement to be executed in the body of the scope of the with.

public interface **com.bea.language.codegen.IRWhileStatement** implements com.bea.language.codegen.IRStatement

Methods **public com.bea.language.codegen.IRExpression getCondition()**
Returns the loop condition expression.

public com.bea.language.codegen.IRStatement getStatement()
Returns the statement to be executed in the body of the loop.

public interface **com.bea.language.codegen.IRVariableInitializer** implements com.bea.language.codegen.IRNode

Represents the initializer of a variable or an array element. Variable initializers can be either single expressions, if the variable being initialized is not an array or they can contain an array initializer, if the element being initialized is also an array. Variable initializers are not used directly to initialize arrays, top-level array initializers are always IRArrayInitializer s.

If isArray returns true, this variable initializer actually represents an array initializer, and you should call getArrayInitializer otherwise this initializer is just a single expression and you should call getInitializerExpression .

Methods	**public boolean isArray()** Returns true if this initializer is for an array. **public com.bea.language.codegen.IRExpression getInitializerExpression()** Returns the expression for this variable initializer. This is only a valid method to call if this initializer is not an array initializer. **public com.bea.language.codegen.IRArrayInitializer getArrayInitializer()** Returns the array initializer for this variable initializer. This is only a valid method to call if this initializer is an array intializer. **public com.bea.language.codegen.IRVariableInitializer getNext()** Returns the next array initializer in the list. Array initializers use linked lists of variable initializers to store the intialization expressions for each element in the array.

public interface **com.bea.language.codegen.IRUnaryExpression** implements com.bea.language.codegen.IRExpression

Methods	**public int getOpcode()** Returns the opcode for the operator. All the opcodes are defined in IROperator . **public com.bea.language.codegen.IRExpression getFirst()** Returns the expression for the first operand.

public interface **com.bea.language.codegen.IRTypeExpression** implements com.bea.language.codegen.IRExpression

Represents and expression that just wraps a type.

Methods	**public com.bea.language.codegen.IRType getType()** Returns the type for this expression.

public interface **com.bea.language.codegen.IRType** implements com.bea.language.codegen.IRNode

Defines the basic information about a Java type. More complicated types inherit from this class and add extra information. The type code is used to specify either one of the primitive types or a class or array type.

Methods	**public int getTypeCode()** Returns the type code for this object. The possible codes are specified as constants on this interface. There are constants for each of the primitive types, as well as the class and array types. **public IType getInfo()** Returns the resolved type for this type object.
Fields	**public static final VOID** The void type.

public static final BYTE
The byte type.

public static final CHAR
The char type.

public static final SHORT
The short type.

public static final INT
The int type.

public static final LONG
The long type.

public static final FLOAT
The float type.

public static final DOUBLE
The double type.

public static final BOOLEAN
The boolean type.

public static final CLASS
Any class or interface type.

public static final ARRAY
The array type.

public interface **com.bea.language.codegen.IRTryStatement** implements com.bea.language.codegen.IRStatement

Methods

public com.bea.language.codegen.IRBlock getBody()
The body of the try statement, which is the part that is "tried".

public com.bea.language.codegen.IRCatches getCatchClause()
The catch clause.

public com.bea.language.codegen.IRFinally getFinallyClause()
The finally clause.

public interface **com.bea.language.codegen.IRThrowStatement** implements com.bea.language.codegen.IRStatement

Methods

public com.bea.language.codegen.IRExpression getExpression()
Returns the expression that will be evaluated to produce the thrown class type.

public interface **com.bea.language.codegen.IRThisExpression** implements com.bea.language.codegen.IRExpression

Represents a reference to a this pointer. A this expression can be qualified by an arbitrary name.

Methods	**public com.bea.language.codegen.IRName getClassName()** Returns the name of the class whose 'this' is being referenced.

public interface **com.bea.language.codegen.IRTernaryExpression** implements com.bea.language.codegen.IRExpression

Methods	**public int getOpcode()** Returns the opcode for the operator. The opcodes are defined in IROperator .
	public com.bea.language.codegen.IRExpression getFirst() Returns the expression for the first operand.
	public com.bea.language.codegen.IRExpression getSecond() Returns the expression for the second operand.
	public com.bea.language.codegen.IRExpression getThird() Returns the expression for the third operand.

public interface **com.bea.language.codegen.IRSynchronizedStatement** implements com.bea.language.codegen.IRStatement

Methods	**public com.bea.language.codegen.IRExpression getExpression()** Returns the expression to synchronize on.
	public com.bea.language.codegen.IRBlock getBody() Returns the body of the synchronized statement. This must be a block.

public interface **com.bea.language.codegen.IRSwitchStatement** implements com.bea.language.codegen.IRStatement

Methods	**public com.bea.language.codegen.IRExpression getSwitchExpression()** Returns the expression for this switch statement. This is the expression that is evaluated in order to determine which case to execute.
	public com.bea.language.codegen.IRSwitchBlockStatementGroup getSwitchBlocks() Returns a linked list of IRSwitchBlockStatementGroup objects which represent the body of the switch statement.

public interface **com.bea.language.codegen.IRSwitchLabel** implements com.bea.language.codegen.IRNode

Methods	**public com.bea.language.codegen.IRExpression getCaseExpression()** Returns the expression for this case.
	public com.bea.language.codegen.IRSwitchLabel getNext() Returns the next label in the list. Switch labels are chained together in a linked list when the appear contiguously in source code.

public interface **com.bea.language.codegen.IRSwitchBlockStatementGroup** implements com.bea.language.codegen.IRNode

Represents a block of labels and statments that make up the body of a switch statement. A single switch block statement group represents a contiguous set of labels followed by the statements to be executed if any of those labels are matched. For example, the following:

case a: case b: case c: System.out.println("foo"); break;

is one switch block statement group where a, b and c are the labels and the println and the break are the statements.

Methods	**public com.bea.language.codegen.IRSwitchLabel getLabels()** Returns the list of labels.
	public com.bea.language.codegen.IRStatement getStatements() Returns the list of statements.
	public com.bea.language.codegen.IRSwitchBlockStatementGroup getNext() Returns the next group in the list. Switch block statement groups are chained together in linked lists to form the body of a switch statement.

public interface **com.bea.language.codegen.IRStatementExpression** implements com.bea.language.codegen.IRStatement

Represents an expression which is being used in a statement. there are a limited number of expressions that can be legally used as statements including:

Assignments
Pre/Post increments/decrements
Method invocations
Class instance creations

Methods	**public com.bea.language.codegen.IRExpression getExpression()** Returns the expression used as a statement.

public interface **com.bea.language.codegen.IRStatement** implements

com.bea.language.codegen.IRNode

Methods **public com.bea.language.codegen.IRStatement getNext()**
Returns the next statement in the list. Statements are chained together in a linked list in blocks.

public interface **com.bea.language.codegen.IRReturnStatement** implements com.bea.language.codegen.IRStatement

Methods **public com.bea.language.codegen.IRExpression getExpression()**
Returns the expression that will produce the return value.

public interface **com.bea.language.codegen.IRParenthesizedExpression** implements com.bea.language.codegen.IRExpression

Methods **public com.bea.language.codegen.IRExpression getExpression()**
Returns the expression.

public interface **com.bea.language.codegen.IROperator** implements com.bea.language.codegen.IRNode

Fields **public static final EQ**
The = operator.

public static final PLUSEQ
The += operator.

public static final MINUSEQ
The -= operator.

public static final MULTEQ
The *= operator.

public static final DIVEQ
The /= operator.

public static final ANDEQ
The <code>&=</code> operator.

public static final OREQ
The |= operator.

public static final XOREQ
The ^= operator.

public static final MODEQ
The %= operator.

public static final LSHIFTEQ
The <code>&lt;&lt;<=</code> operator.

public static final RSHIFTEQ
The = operator.

public static final URSHIFTEQ
The = operator.

public static final OROR
The || operator.

public static final ANDAND
The <code>&&</code> operator.

public static final OR
The | operator.

public static final XOR
The ^ operator.

public static final AND
The <code>&</code> operator.

public static final EQEQ
The == operator.

public static final NOTEQ
The != operator.

public static final LT
The <operator.

public static final GT
The >operator.

public static final LTEQ
The <= operator.

public static final GTEQ
The >= operator.

public static final LSHIFT
The operator.

public static final RSHIFT
The operator.

public static final URSHIFT
The operator.

public static final PLUS
The + operator.

public static final MINUS
The - operator.

public static final MULT
The * operator.

public static final DIV
The / operator.

public static final MOD
The % operator.

public static final PREPLUSPLUS
The prefix ++ operator.

public static final PREMINUSMINUS
The prefix -- operator.

public static final POSTPLUSPLUS
The postfix ++ operator.

public static final POSTMINUSMINUS
The postfix -- operator.

public static final NOT
The ! operator.

public static final COMP
The ~ operator.

public static final CONDITIONAL
The ?: operator.

public static final INSTANCEOF
The instanceof operator.

public static final EQEQEQ
The === operator strict comparision for XScript.

public static final NOTEQEQ
The !== operator strict comparision for XScript.

public static final IN
The in operator used in for/in for XScript.

public static final TYPEOF
The typeof operator used in XScript.

public interface **com.bea.language.codegen.IRNode**

The base class for all nodes in the intermediate representation. Every node has a unique node type code. All the possible node types are defined as members of this interface.

Methods

public int getNodeType()
Returns the node type code. This uniquely identifies a node.

public IToken getFirstToken()
Returns the start of the source range of this node or null.

public IToken getLastToken()
Returns the end of the source range of this node or null.

Fields

public static final LITERAL

public static final TYPE

public static final CLASSORINTERFACETYPE

public static final ARRAYTYPE

public static final NAME

public static final COMPILATIONUNIT

public static final IMPORTDECLARATION

public static final MODIFIERS

public static final CLASSDECLARATION

public static final CLASSBODYDECLARATION

public static final FIELDDECLARATION

public static final METHODDECLARATION

public static final FORMALPARAMETER

public static final INITIALIZERBLOCK

public static final CONSTRUCTORDECLARATION

public static final CONSTRUCTORINVOCATION

public static final ARRAYINITIALIZER

public static final VARIABLEINITIALIZER

public static final BLOCK

public static final CLASSDECLARATIONSTATEMENT

public static final LOCALVARIABLEDECLARATIONSTATEMENT

public static final LOCALVARIABLEDECLARATION

public static final STATEMENT

public static final EMPTYSTATEMENT

public static final LABELEDSTATEMENT

public static final STATEMENTEXPRESSION

public static final IFTHENELSESTATEMENT

public static final SWITCHSTATEMENT

public static final SWITCHBLOCKSTATEMENTGROUP

public static final SWITCHLABEL

public static final WHILESTATEMENT

public static final DOSTATEMENT

```
public static final FORSTATEMENT
public static final IDENTIFIEROPT
public static final BREAKSTATEMENT
public static final CONTINUESTATEMENT
public static final RETURNSTATEMENT
public static final THROWSTATEMENT
public static final SYNCHRONIZEDSTATEMENT
public static final TRYSTATEMENT
public static final CATCHES
public static final FINALLY
public static final ASSERTSTATEMENT
public static final PARENTHESIZEDEXPRESSION
public static final CLASSLITERAL
public static final THISEXPRESSION
public static final CLASSINSTANCECREATIONEXPRESSION
public static final ARRAYCREATIONEXPRESSION
public static final DIMS
public static final FIELDACCESS
public static final METHODINVOCATION
public static final ARRAYACCESS
public static final CASTEXPRESSION
public static final ASSIGNMENTOPERATOR
public static final EXPRESSION
public static final UNARYEXPRESSION
public static final BINARYEXPRESSION
public static final TERNARYEXPRESSION
public static final NAMEEXPRESSION
public static final TYPEEXPRESSION
public static final OPERATOR
public static final ANNOTATION
public static final ANNOTATIONATTRIBUTE
```

public static final FORINSTATEMENT

public static final DELETESTATEMENT

public static final METHODDECLARATIONSTATEMENT

public static final WITHSTATEMENT

public static final XMLDOTQUERY

public static final XMLEXPRESSION

public static final NAMESPACESTATEMENT

public static final XMLLISTEXPRESSION

public static final XQRLSTATEMENT

public interface **com.bea.language.codegen.IRNamespaceStatement** implements com.bea.language.codegen.IRStatement

Methods

public java.lang.String getName()
Returns variable of type namespace.

public com.bea.language.codegen.IRExpression getExpression()
Returns the expression that is the namespace's URI.

public interface **com.bea.language.codegen.IRNameExpression** implements com.bea.language.codegen.IRExpression

Represents and expression that just contains a name.

Methods

public com.bea.language.codegen.IRName getName()
Returns the name for this expression.

public com.bea.language.info.ILangVariable getVariable()
Returns the variable being referenced by this name.

public interface **com.bea.language.codegen.IRName** implements com.bea.language.codegen.IRNode

Represents simple or qualified name. If the name is qualified, each simple component is represented by an <code>IRName</code>. The next component can be retrieved by using <code>getNext()</code>. Names are stored in reverse order, that is com.bea.language would be represented as language->bea->com. <p> Also each name part can include information about what the name represents. This should be present on any name that is used in a context that allows multiple parts. Single-part names (for example, the IDs of fields and methods) will not have this information.

Methods

public java.lang.String getID()
Returns the identifier for this part of the name.

public com.bea.language.codegen.IRName.INameInfo getInfo()
Returns the info for this part of the name or null if not available.

public com.bea.language.codegen.IRName getNext()
Returns the next component of this name.

public static interface **com.bea.language.codegen.IRName.INameInfo**

Defines the type of info available on names.

Methods

public int getType()
Returns the type of this name.

public java.lang.Object get()
Returns information about the object or type of object being referenced by this name. The type indicates what sort of object will be returned.

Fields

public static final TYPE_PACKAGE
Indicates that this name represents a package. The object returned will be an instanceof IPackage .

public static final TYPE_USERTYPE
Indicates that this name represents a class or interface. The object returned will be an instance of IReferenceType

public static final TYPE_VARIABLE
Indicates that this name represents a field, argument, or local variable. The object returned will be an instance of ILangVariable .

public static final TYPE_METHOD
Indicates that this name represents a method. The object returned will be an instanceof IMethodLocation .

public interface **com.bea.language.codegen.IRModifiers** implements com.bea.language.codegen.IRNode

Methods

public int getModifierBits()
The integer which holds the bits for the modifier set.

Fields

public static final PUBLIC
The public modifier.

public static final PROTECTED
The protected modifier.

public static final PRIVATE
The private modifier.

public static final STATIC
The static modifier.

public static final ABSTRACT
The abstract modifier.

public static final FINAL
The final modifier.

public static final NATIVE
The native modifier.

public static final SYNCHRONIZED
The synchronized modifier.

public static final TRANSIENT
The transient modifier.

public static final VOLATILE
The volatile modifier.

public static final STRICTFP
The strictfp modifier.

public interface **com.bea.language.codegen.IRMethodInvocation** implements com.bea.language.codegen.IRExpression

Methods

public boolean isSuper()
Returns true if there is a reference to super in this invocation.

public com.bea.language.codegen.IRName getClassName()
For super invocations, returns the name of the class the method is being invoked. For non-super invocations, this returns null.

public com.bea.language.codegen.IRExpression getClassExpression()
Returns the expression on which this method is being invoked or null if no such expression exists.

public com.bea.language.codegen.IRName getMethodName()
Returns the method name.

public com.bea.language.codegen.IRExpression getArguments()
Returns the linked list of argument expressions.

public IMethod getTarget()
Returns the target method of this call.

public interface **com.bea.language.codegen.IRMethodDeclarationStatement** implements com.bea.language.codegen.IRStatement

Methods

public com.bea.language.codegen.IRMethodDeclaration getMethodDeclaration()
Returns the method declaration.

public interface **com.bea.language.codegen.IRMethodDeclaration** implements com.bea.language.codegen.IRClassBodyDeclaration

Methods	**public com.bea.language.codegen.IRName getName()** Returns the name of the method.
	public com.bea.language.codegen.IRModifiers getModifiers() Returns the modifiers on the method.
	public com.bea.language.codegen.IRType getReturnType() Returns the return type of the method.
	public com.bea.language.codegen.IRFormalParameter getParameters() Returns the list of formal parameters for the method.
	public com.bea.language.codegen.IRClassOrInterfaceType getThrowList() Returns the list of exception types the method can throw.
	public com.bea.language.codegen.IRAnnotation getAnnotations() Returns the root node of the annotation tree.
	public com.bea.language.codegen.IRBlock getBody() Returns the body of this method, which must be a block.
	public IMethod getTypeSignature() Returns the resolved type signature of this method.
	public boolean isUsedExternally() Determines whether this method is used in another class of the file.

public interface **com.bea.language.codegen.IRLocalVariableDeclarationStatement** implements com.bea.language.codegen.IRStatement

Methods	**public com.bea.language.codegen.IRLocalVariableDeclaration getDeclarations()** Returns the list of local variable declarations for this statment.

public interface **com.bea.language.codegen.IRLocalVariableDeclaration** implements com.bea.language.codegen.IRNode, com.bea.language.info.ILangVariable

Methods	**public java.lang.String getName()** Returns the name of this local variable declaration.
	public com.bea.language.codegen.IRType getVarType() Returns the type of this local variable declaration.

public IType getType()
Returns the resolved type of the declaration.

public com.bea.language.codegen.IRModifiers getModifiers()
Returns the modifiers for this local variable declaration.

public com.bea.language.codegen.IRVariableInitializer getInitializer()
Returns the initializer for this local variable declaration.

public boolean isVar()
Return is variable is marked as var (XScript)

public com.bea.language.codegen.IRLocalVariableDeclaration getNext()
Returns the next local variable declaration in this list. Local variable declarations are chained together in a linked list whenever they appear on one line and share a type, as in:

int i, j, k;

public interface **com.bea.language.codegen.IRLiteral** implements com.bea.language.codegen.IRExpression

A literal represents a literal value that can appear in the source code such as "foo" or 1.0e+123 . A literal consists of a type code, to identify the type of the literal and a string value.

Methods

public int getTypeCode()
Returns the type code for this literal. The possible type codes are defined as static constants.

public java.lang.String getValue()
Returns the string value of this literal.

Fields

public static final INTEGER
An integer value.

public static final FLOAT
A floating point (float or double) value.

public static final BOOLEAN
A boolean value.

public static final CHARACTER
A character value.

public static final STRING
A string value.

public static final XML
An XML value.

public static final XMLLIST
An XML value.

public static final VARIANT
A Variant value used by XScript for Undefined, NaN, Infinity.

public static final NULL
A null value.

public interface **com.bea.language.codegen.IRLabeledStatement** implements com.bea.language.codegen.IRStatement

Represents a statement that has a label next to it.

Methods	**public java.lang.String getLabel()** Returns the text of the label. **public com.bea.language.codegen.IRStatement getStatement()** Returns the statement after the label.

public interface **com.bea.language.codegen.IRInitializerBlock** implements com.bea.language.codegen.IRClassBodyDeclaration

Methods	**public boolean isStatic()** Returns true if this is a static initializer. **public com.bea.language.codegen.IRBlock getBody()** Returns the body of the initializer, which must be a block.

public interface **com.bea.language.codegen.IRIfThenElseStatement** implements com.bea.language.codegen.IRStatement

Methods	**public com.bea.language.codegen.IRExpression getCondition()** Returns the test condition for the if . **public com.bea.language.codegen.IRStatement getTrueStatement()** Returns the statement to execute if the condition is true. **public com.bea.language.codegen.IRStatement getFalseStatement()** Returns the statement to execute if the condition is false.

public interface **com.bea.language.codegen.IRForStatement** implements com.bea.language.codegen.IRStatement

Methods	**public com.bea.language.codegen.IRStatement getInitStatement()** Returns the statement used to initialize the for loop. **public com.bea.language.codegen.IRExpression getTestExpression()** Returns the test expression for the loop. **public com.bea.language.codegen.IRStatement getUpdateStatement()** Returns the statement used to iterate the loop.

public com.bea.language.codegen.IRExpression getEnumeratorVariable()
Returns the lhs of the IN operator for/in in XScript.

public com.bea.language.codegen.IRExpression getEnumExpression()
Returns the expression to enumerate on for the IN operator for/in in XScript

public com.bea.language.codegen.IRStatement getBody()
Returns the statement executed in the body of the loop.

public interface **com.bea.language.codegen.IRFormalParameter** implements com.bea.language.codegen.IRNode, com.bea.language.info.ILangVariable

Methods

public java.lang.String getName()
Returns the name of this formal parameter.

public com.bea.language.codegen.IRType getParamType()
Returns the type of this formal parameter.

public IType getType()
Returns the resolved type of this formal parameter.

public com.bea.language.codegen.IRModifiers getModifiers()
Returns the modifiers object for this formal paramter.

public com.bea.language.codegen.IRFormalParameter getNext()
Returns the next formal parameter in the list. This is used to chain together the formal parameters for a method into a linked list.

public interface **com.bea.language.codegen.IRFinally** implements com.bea.language.codegen.IRNode

Methods

public com.bea.language.codegen.IRBlock getBody()
Returns the body of this finally clause.

public interface **com.bea.language.codegen.IRFieldDeclaration** implements com.bea.language.codegen.IRClassBodyDeclaration

Methods

public java.lang.String getName()
Returns the name of this field.

public com.bea.language.codegen.IRModifiers getModifiers()
Returns the modifiers for this field.

public com.bea.language.codegen.IRType getType()
Returns the type of this field.

public com.bea.language.codegen.IRVariableInitializer getInitializer()

Returns the initializer for this field.

public com.bea.language.codegen.IRAnnotation getAnnotations()
Returns the root node of the annotation tree.

public IField getTypeSignature()
Returns the resolved type signature for this field.

public boolean isUsedExternally()
Determines whether this field is used in another class of the file.

public interface **com.bea.language.codegen.IRFieldAccess** implements com.bea.language.codegen.IRExpression

Represents a field access. There are few wrinkes here that are worth mentioning.

Field accesses can refer to the super class in which case isSuper will be true.
Depending on the situation, the class on which the field is being accessed can either be represented by a name or an expression. For instance new Foo().bar is a field access using an expression as the class name.

Methods

public boolean isSuper()
Returns true if the super keyword is used in this expression.

public com.bea.language.codegen.IRName getClassName()
For accesses using 'super', this returns the name of the class on which the field access is occuring.

public com.bea.language.codegen.IRExpression getClassExpression()
For accesses without 'super, this returns the expression that evaluates to the object on which the field is to be accessed.

public java.lang.String getFieldName()
Returns the name of the accessed field.

public IField getTarget()
Returns the target field access.

public class **com.bea.language.codegen.IRFactory**

Constructors

public IRFactory()

Methods

public static com.bea.language.codegen.IIRFactory get()

**public static void set(
IIRFactory factory)**

public interface **com.bea.language.codegen.IRExpression** implements com.bea.language.codegen.IRNode

Methods

public IType getCheckedType()
Returns the type checked type of this expression.

public boolean isConstant()
Determines whether this expression has a constant value.

public com.bea.language.info.IConstant getConstant()
Returns the constant value of this expression if constant. This will throw an exception if the expression is not constant.

public com.bea.language.codegen.IRExpression getNext()
Returns the next expression in the list. There are too many cases to enumerate them all.

public interface **com.bea.language.codegen.IREmptyStatement** implements com.bea.language.codegen.IRStatement

Empty statements represent the case where a ; is used as a statement. They do nothing and are simply placeholders.

public interface **com.bea.language.codegen.IRDoStatement** implements com.bea.language.codegen.IRStatement

Methods

public com.bea.language.codegen.IRExpression getCondition()
Returns the loop condition expression.

public com.bea.language.codegen.IRStatement getStatement()
Returns the statement to be executed in the body of the loop.

public interface **com.bea.language.codegen.IRDeleteStatement** implements com.bea.language.codegen.IRStatement

Methods

public com.bea.language.codegen.IRExpression getExpression()
Returns the expression that will be evaluated and deleted

public interface **com.bea.language.codegen.IRContinueStatement** implements com.bea.language.codegen.IRStatement

Methods

public java.lang.String getLabel()
Returns the label for the continue statement.

public com.bea.language.codegen.IRStatement getTarget()

Returns the target statement to break/continue at, if labeled it's the labeled statement.

public interface **com.bea.language.codegen.IRConstructorInvocation** implements com.bea.language.codegen.IRExpression

Methods

public com.bea.language.codegen.IRExpression getClassExpression()
Returns the expression for the class on which this constructor is being invoked, if one exists.

public boolean isSuper()
Returns true if there is a reference to super in this invocation.

public IConstructor getTarget()
Returns the target constructor of this call.

public com.bea.language.codegen.IRExpression getArguments()
Returns the linked list of argument expressions.

public interface **com.bea.language.codegen.IRConstructorDeclaration** implements com.bea.language.codegen.IRClassBodyDeclaration

Methods

public com.bea.language.codegen.IRName getName()
Returns the name of the constructor. This should match the class name in which this constructor appears.

public com.bea.language.codegen.IRModifiers getModifiers()
Returns the modifiers for the constructor.

public com.bea.language.codegen.IRFormalParameter getParameters()
Returns the list of formal parameters for the constructor.

public com.bea.language.codegen.IRClassOrInterfaceType getThrowList()
The list of exception types this constructor can throw.

public com.bea.language.codegen.IRAnnotation getAnnotations()
Returns the root node of the annotation tree.

public com.bea.language.codegen.IRBlock getBody()
Returns the body of the constructor, which must be a block.

public IConstructor getTypeSignature()
Returns the resolved type signature for the constructor.

public boolean isUsedExternally()
Determines whether this is used by another class of the file.

public interface **com.bea.language.codegen.IRCompilationUnit** implements com.bea.language.codegen.IRNode

Methods **public com.bea.language.codegen.IRName getPackage()**
Returns the package declaration for this file.

public com.bea.language.codegen.IRClassDeclaration getClasses()
Returns the list of classes defined in this file.

public interface **com.bea.language.codegen.IRClassOrInterfaceType** implements com.bea.language.codegen.IRType

Represents a class or interface. It should be noted that the names of these types are not resolved. If you want those you should get the IReferenceType object for the type.

Methods **public com.bea.language.codegen.IRName getName()**
Returns the name of this type.

public com.bea.language.codegen.IRClassOrInterfaceType getNext()
Returns the next type in the list. These types are chained into a linked list in three cases:

In the list of interfaces that a class implements.

In the list of interfaces that another interface extends.

In the list of types on a throws clause of a method.

public interface **com.bea.language.codegen.IRClassLiteral** implements com.bea.language.codegen.IRExpression

Represents literal references to the class object for a particular type. An example would be Object.class .

Methods **public com.bea.language.codegen.IRType getClassType()**
Returns the type whose class is being retrieved.

public interface **com.bea.language.codegen.IRClassInstanceCreationExpression** implements com.bea.language.codegen.IRExpression

Represents a class creation expression, which is basically an expression that up a class. There are a two special things that can appear as part of one of these expressions.

First, a new expression may be qualified by a type, which is used to create inner classes outside of the class in which they are defined. This qualifier appears right before the new operator.
Second, a new expression may have a "body" when an anonymous class is being created. This body will be a list of class body declarations which are used to extend the base class.

Methods **public com.bea.language.codegen.IRExpression getQualifier()**
Returns the qualifier before the new operator.

public com.bea.language.codegen.IRType getType()
Returns the type of the class (or base class) that is being created.

public com.bea.language.codegen.IRExpression getArguments()
Returns the list of expressions that will be the arguments to the constructor.

public boolean hasBody()
Determines whether this instance has a body (i.e., this is an instantiation of an anonymous class).

public com.bea.language.codegen.IRClassBodyDeclaration getBodyDeclarations()
Returns the class body declarations that are present if this is the declaration of an anonymous class.

public java.util.Collection getReferencedVariables()
Returns a collection of variables from immediately enclosing statement blocks that were referenced inside this anonymous class or null if no local variables were referenced or this expression has no body.

public IConstructor getConstructor()
Returns the constructor that is being called for this expression.

public interface **com.bea.language.codegen.IRClassDeclarationStatement** implements com.bea.language.codegen.IRStatement

Methods

public com.bea.language.codegen.IRClassDeclaration getClassDeclaration()
Returns the class declaration.

public interface **com.bea.language.codegen.IRClassDeclaration** implements com.bea.language.codegen.IRClassBodyDeclaration

Represents a class or interface declaration.

Methods

public java.lang.String getName()
Returns the name of this class.

public boolean isInterface()
Returns true if this declaration represents an interface.

public com.bea.language.codegen.IRModifiers getModifiers()
Returns the modifiers for this declaration.

public com.bea.language.codegen.IRType getSuperClassType()
Returns the list of super classes for this declaration.

public com.bea.language.codegen.IRType getInterfaceTypes()
Returns the list of super interfaces for this declaration.

public com.bea.language.codegen.IRAnnotation getAnnotations()
Returns the root node of the annotation tree.

public com.bea.language.codegen.IRClassBodyDeclaration getBodyDeclarations()

Returns the declarations in the body of this declaration.

public IReferenceType getTypeSignature()
Returns the resolved type signature for this class declaration.

public java.util.Collection getReferencedVariables()
Returns a collection of variables from immediately enclosing statement blocks that were referenced inside this anonymous class or null if no local variables were referenced.

public interface **com.bea.language.codegen.IRClassBodyDeclaration** implements com.bea.language.codegen.IRNode

Represents any declaration that can appear directly in a class. This includes fields, methods, classes, interfaces, constructors, and initializers.

Methods

public com.bea.language.codegen.IRClassBodyDeclaration getNext()
Returns the next class body declaration in the list. Class body declarations are chained in a linked list to define the definitions inside a class body.

public interface **com.bea.language.codegen.IRCatches** implements com.bea.language.codegen.IRNode

Methods

public com.bea.language.codegen.IRFormalParameter getParameter()
Returns the single formal parameter for this catch clause.

public com.bea.language.codegen.IRBlock getBody()
Gets the body of this catch clause, which must be a block.

public com.bea.language.codegen.IRCatches getNext()
Returns the next catch clause in the list. Catch clauses are chained together in this list when they appear together in the source code.

public IReferenceType getResolvedType()
Returns the resolved type of the class being caught.

public interface **com.bea.language.codegen.IRCastExpression** implements com.bea.language.codegen.IRExpression

Methods

public com.bea.language.codegen.IRExpression getTypeExpression()
Returns the expression defining the type to cast to.

public com.bea.language.codegen.IRExpression getExpression()
Returns the expression defining the object to cast.

public interface **com.bea.language.codegen.IRBreakStatement** implements

com.bea.language.codegen.IRStatement

Methods

public java.lang.String getLabel()
Returns the label for the break statement.

public com.bea.language.codegen.IRStatement getTarget()
Returns the target statement to break/continue at, if labeled it's the labeled statement.

public interface **com.bea.language.codegen.IRBlock** implements com.bea.language.codegen.IRStatement

Methods

public com.bea.language.codegen.IRStatement getStatementList()
Returns the list of statements in the block.

public interface **com.bea.language.codegen.IRBinaryExpression** implements com.bea.language.codegen.IRExpression

Methods

public int getOpcode()
Returns the opcode for the operator. The opcodes are defined in IROperator .

public com.bea.language.codegen.IRExpression getFirst()
Returns the expression for the first operand.

public com.bea.language.codegen.IRExpression getSecond()
Returns the expression for the second operand.

public interface **com.bea.language.codegen.IRAssertStatement** implements com.bea.language.codegen.IRStatement

Represents an assert statement in the source code.

Methods

public com.bea.language.codegen.IRExpression getCondition()
Returns the test condition for the assert .

public com.bea.language.codegen.IRExpression getDetail()
Returns the detail message for the assert .

public interface **com.bea.language.codegen.IRArrayType** implements com.bea.language.codegen.IRType

Methods

public com.bea.language.codegen.IRType getElementType()

Returns the type of an element in this array. This type is always a primitive type or a class type. So for an array of type int[][][][] the returned type will still be int .

public int getDimension()
Returns the dimensionality of this array.

public interface **com.bea.language.codegen.IRArrayInitializer** implements com.bea.language.codegen.IRNode

Represents the initializer of an array, or an element of an array that is also an array. Array initializers are composed of a linked list of IRVariableInitializer s.

Methods

public com.bea.language.codegen.IRVariableInitializer getInitializerList()
Returns the list of variable initializers for each of the elements in this array initializer. These inidividual initializers can be arrays themselves.

public interface **com.bea.language.codegen.IRArrayCreationExpression** implements com.bea.language.codegen.IRExpression

Represents an expression that creates a new instance of an array. All such valid expressions will satisfy these invariant that the number of dimensions in the type of the array expression itself is equal to the number of dimension expressions plus the number of extra dimensions.

Methods

public com.bea.language.codegen.IRType getType()
Returns the element type of the array.

public com.bea.language.codegen.IRExpression getDimensionExpressions()
Returns the list expressions used to size the array.

public int getExtraDimensions()
Returns the number of dimensions that do not have sizes.

public com.bea.language.codegen.IRArrayInitializer getInitializer()
Returns the array initializer for this new array.

public interface **com.bea.language.codegen.IRArrayAccess** implements com.bea.language.codegen.IRExpression

Methods

public com.bea.language.codegen.IRExpression getArrayExpression()
Returns the expression used to specify the array.

public com.bea.language.codegen.IRExpression getSubscriptExpression()
Returns the subscript expression.

public interface **com.bea.language.codegen.IRAnnotationAttribute**

implements com.bea.language.codegen.IRNode

Describes an attribute of an annotation on an declared entity in the file. An annotation attribute is similar to an attribute in XML.

Methods

public java.lang.String getName()
Returns the name of this attribute, which may include a ':'.

public java.lang.Object getValue()
Returns the value of this attribute.

public java.lang.String getText()
Returns the original text value of this attribute.

public IAnnotation getTypeSignature()

public com.bea.language.codegen.IRAnnotationAttribute getNextSibling()
Returns the next attribute on this annotation.

public interface **com.bea.language.codegen.IRAnnotation** implements com.bea.language.codegen.IRNode

Describes an annotation on a declared entity in the file. An annotation is similar to a tag in XML.

Methods

public java.lang.String getName()
Returns the name of this annotation, which may include a ':'.

public com.bea.language.codegen.IRAnnotationAttribute getFirstAttribute()
Returns the first attribute of this annotation.

public IAnnotation getTypeSignature()
Returns the resolved shape of this annotation.

public com.bea.language.codegen.IRAnnotation getFirstChild()
Returns the first child of this annotation.

public com.bea.language.codegen.IRAnnotation getNextSibling()
Returns the next sibling of this annotation.

Fields

public static final ROOT_NAME
Defines the name of the root node in the annotation tree. This contains the ' and ' characters which are illegal in normal tag names.

public interface **com.bea.language.codegen.IIRFactory**

Methods

public com.bea.language.codegen.IRAnnotation createAnnotation(
String name,
IRAnnotationAttribute attrs,
IRAnnotation children,
IAnnotation signature,
IRAnnotation next)
Creates an annotation with the properties. The attributes and children must be already be linked together with the given references being to the head of the lists.

public com.bea.language.codegen.IRAnnotation createAnnotation(
String name,
Collection attrs,
Collection children,
IAnnotation signature,
IRAnnotation next)

Creates an annotation with the given properties. The attributes and children will be linked together in the given order. Note that all of these attributes must have been created with this factory.

public com.bea.language.codegen.IRAnnotation createAnnotationRoot(
IRAnnotation children)

Creates the root of the annotation root for any entity.

public com.bea.language.codegen.IRAnnotationAttribute createAnnotationAttribute(
String name,
Object value,
IAnnotation signature,
IRAnnotationAttribute next)

Creates an attribute with the same text and value.

public com.bea.language.codegen.IRAnnotationAttribute createAnnotationAttribute(
String name,
Object value,
String text,
IAnnotation signature,
IRAnnotationAttribute next)

Creates an attribute with different text and value.

public com.bea.language.codegen.IRExpression createArguments(
Collection arguments)

Converts the given list of arguments into linked form. It is required that each such argument have been created by this factory.

public com.bea.language.codegen.IRArrayAccess createArrayAccess(
IRExpression arrayExpr,
IRExpression subscriptExpr)

public com.bea.language.codegen.IRArrayCreationExpression createArrayCreationExpression(
IRType elemType,
IRExpression dimExpr,
int extraDims,
IRArrayInitializer initializer)

Creates an array with the given properties.

Parameters

elemType - Indicates the element type of the array. This must be a primitive type, class, or interface.

dimExpr - Provides the expressions for the sizes of the array dimensions.

extraDims - Counts the number of additional array dimensions for which sizes were not provided.

initializer - Records the expression for initializing the elements of the array or is null indicating that default initial values are to be used.

public com.bea.language.codegen.IRArrayInitializer createArrayInitializer(
IRVariableInitializer initList,
IType info)

public com.bea.language.codegen.IRArrayType createArrayType(
IRType elementType,
int dimension,

IType info)

public com.bea.language.codegen.IRBinaryExpression createBinaryExpression(
int opcode,
IRExpression first,
IRExpression second)

public com.bea.language.codegen.IRBlock createBlock(
IRStatement statements)
Creates a block from a list of statements that are already linked.

public com.bea.language.codegen.IRBlock createBlock(
Collection statements)
Creates a block from the given list of statements. It is required that all of the given statements were created from this factory.

public com.bea.language.codegen.IRBreakStatement createBreakStatement(
String label)

public com.bea.language.codegen.IRCastExpression createCastExpression(
IRTypeExpression typeExpr,
IRExpression expr)

public com.bea.language.codegen.IRCatches createCatches(
IRFormalParameter param,
IRBlock body,
IRCatches next,
IReferenceType type)

public com.bea.language.codegen.IRClassBodyDeclaration createClassBodyDeclarations(
Collection decls)
Creates a linked list of declarations from the given list. It is required that all of the given declarations were created from this factory.

public com.bea.language.codegen.IRClassDeclaration createClassDeclaration(
String name,
boolean isInterface,
IRModifiers mods,
IRType superClass,
IRType interTypes,
IRAnnotation annotations,
IRClassBodyDeclaration bodyDecls,
IReferenceType signature,
IRClassBodyDeclaration next)

public com.bea.language.codegen.IRClassDeclarationStatement createDeclarationStatement(
IRClassDeclaration declaration)

public com.bea.language.codegen.IRClassInstanceCreationExpression createClassInstanceCreationExpression(
IRType type,
IRExpression arguments,
IConstructor constructor)
Creates an instance of the given class, passing the given arguments to the constructor.

public com.bea.language.codegen.IRClassInstanceCreationExpression createClassInstanceCreationExpression(
IRExpression qualifier,
IRType type,
IRExpression arguments,

IConstructor constructor)
Creates an instance of the member type with the given name of the type of the qualifier expression. For example: A a = ...; a.new B(); This will create an instance of class A.B , with a being the enclosing instance. Note that, in this case, the type must consist of a single identifier.

public com.bea.language.codegen.IRClassInstanceCreationExpression createClassInstanceCreationExpression(
IRType type,
IRExpression arguments,
IConstructor constructor,
IRClassBodyDeclaration decls)
Creates an instance of an anonymous subclass of the given class (or an anonymous implementor of the given interface).

public com.bea.language.codegen.IRClassInstanceCreationExpression createClassInstanceCreationExpression(
IRExpression qualifier,
IRType type,
IRExpression arguments,
IConstructor constructor,
IRClassBodyDeclaration decls)
Creates an instance of an anonymous subclass of the given member type of the type of the qualifying expression. The result of evaluating the qualifying expression will be the enclosing instance.

public com.bea.language.codegen.IRClassLiteral createClassLiteral(
IRType classType)

public com.bea.language.codegen.IRClassOrInterfaceType createClassOrInterfaceType(
IRName name,
IRClassOrInterfaceType next,
IType type)

public com.bea.language.codegen.IRConstructorDeclaration createConstructorDeclaration(
IRName name,
IRModifiers modifiers,
IRFormalParameter param,
IRClassOrInterfaceType throwList,
IRAnnotation annotations,
IRBlock body,
IConstructor typeSignature,
IRClassBodyDeclaration decl)

public com.bea.language.codegen.IRConstructorInvocation createConstructorInvocation(
boolean isSuper,
IRExpression classExpr,
IRExpression arguments,
IConstructor target)
Creates a call to a constructor of this class or the superclass. This is only legal as the first statement of a constructor.
Parameters
isSuper - Determines whether this is a call to a constructor in the superclass or this class.
classExpr - An expression for the outer class required by the superclass or null if this is not a super call or if the superclass does not require a different outer class.
arguments - The arguments to the constructor.
target - The type info for the constructor being called.

public com.bea.language.codegen.IRContinueStatement createContinueStatement(
String label)

public com.bea.language.codegen.IRDoStatement createDoStatement(
IRExpression condition,
IRStatement statement)

public com.bea.language.codegen.IREmptyStatement createEmptyStatement()

public com.bea.language.codegen.IRExpression createExpression(
IType type,
IRExpression next)

public com.bea.language.codegen.IRFieldAccess createFieldAccess(
IRExpression classExpression,
String fieldName,
IField target)
Creates an access to the given field on the result of the given expression.

public com.bea.language.codegen.IRFieldAccess createFieldAccess(
IRName className,
String fieldName,
IField target)
Creates an access to a field in a superclass. The classType is optional, and if present, indicates the name of the superclass in which the field exists.

public com.bea.language.codegen.IRFieldAccess createFieldAccess(
boolean isSuper,
IRName className,
IRExpression classExpression,
String fieldName,
IField target)
This version of createFieldAccess takes all possible parameters and is mainly useful when copying another field access.

public com.bea.language.codegen.IRFieldDeclaration createFieldDeclaration(
String name,
IRModifiers modifiers,
IRType type,
IRVariableInitializer initializer,
IRAnnotation annotations,
IField signature,
IRClassBodyDeclaration next)

public com.bea.language.codegen.IRFinally createFinally(
IRBlock body)

public com.bea.language.codegen.IRFormalParameter createFormalParameter(
String name,
IRType paramType,
IType type,
IRModifiers modifiers,
IRFormalParameter next)

public com.bea.language.codegen.IRForStatement createForStatement(
IRStatement init,
IRExpression test,
IRStatement update,
IRStatement body)

public com.bea.language.codegen.IRIfThenElseStatement createIfThenElseStatement(
IRExpression condition,

IRStatement trueStatement,
IRStatement falseStatement)

public com.bea.language.codegen.IRInitializerBlock createInitializerBlock(
boolean isStatic,
IRBlock body,
IRClassBodyDeclaration next)

public com.bea.language.codegen.IRLabeledStatement createLabeledStatement(
String label,
IRStatement statement)

public com.bea.language.codegen.IRLiteral createLiteral(
int value)
Creates an integer literal with the given value.

public com.bea.language.codegen.IRLiteral createLiteral(
long value)
Creates a long literal with the given value.

public com.bea.language.codegen.IRLiteral createLiteral(
float value)
Creates a float literal with the given value.

public com.bea.language.codegen.IRLiteral createLiteral(
double value)
Creates a double literal with the given value.

public com.bea.language.codegen.IRLiteral createLiteral(
boolean value)
Creates a boolean literal with the given value.

public com.bea.language.codegen.IRLiteral createLiteral(
char value)
Creates a character literal with the given value.

public com.bea.language.codegen.IRLiteral createLiteral(
String value)
Creates a string literal with the given value.

public com.bea.language.codegen.IRLiteral createNullLiteral()
Creates a null literal.

public com.bea.language.codegen.IRLiteral createLiteral(
IType type,
int typeCode,
String value,
IConstant constValue)
Creates a literal with the given value. This should only be used when copying a literal.

public com.bea.language.codegen.IRLocalVariableDeclaration createLocalVariableDeclaration(
String name,
IRType varType,
IType type,
IRModifiers modifiers,
IRVariableInitializer init,
IRLocalVariableDeclaration next,
boolean fVarEncountered)

public com.bea.language.codegen.IRLocalVariableDeclarationStatement createLocalVariableDeclarationStatement(

IRLocalVariableDeclaration declaration)

public com.bea.language.codegen.IRMethodDeclaration createMethodDeclaration(
IRName name,
IRModifiers modifiers,
IRType returnType,
IRFormalParameter params,
IRClassOrInterfaceType throwList,
IRAnnotation annotations,
IRBlock body,
IMethod typeSignature,
IRClassBodyDeclaration next)

public com.bea.language.codegen.IRMethodDeclarationStatement createMethodDeclarationStatement(
IRMethodDeclaration methodDeclaration)

public com.bea.language.codegen.IRMethodInvocation createMethodInvocation(
IRName methodName,
IRExpression arguments,
IMethod target)
Creates a method call where a name is used by itself without a preceding expression. Here are some examples: System.out.println("hello") foo() In the latter case, foo could be a method in that class.

public com.bea.language.codegen.IRMethodInvocation createMethodInvocation(
IRExpression precExpr,
String methodName,
IRExpression arguments,
IMethod target)
Creates a method call where the name is preceded by some kind of expression. Here is an example: "foo".substring(1)

public com.bea.language.codegen.IRMethodInvocation createSuperMethodInvocation(
IRName className,
String methodName,
IRExpression arguments,
IMethod target)
Creates a method call where the super operator is used. Here is an example: MyClass.super.foo()

public com.bea.language.codegen.IRMethodInvocation createMethodInvocation(
IRName className,
IRExpression classExpr,
IRName methodName,
boolean isSuper,
IMethod target,
IRExpression arguments)
Creates a method call. This version takes all possible parts that could be used in a method invocation. This should never be called except to copy another method invocation.

public com.bea.language.codegen.IRModifiers createModifiers(
int modifierBits)

public com.bea.language.codegen.IRName createName(
String ident,
IRName qual)
Creates a name with the given ID and qualifier but no info.

public com.bea.language.codegen.IRName createName(
String ident,
IRName.INameInfo info,

IRName qual)
Creates a name with the given ID, qualifier, and info.

public com.bea.language.codegen.IRNameExpression createNameExpression(
IRName name,
ILangVariable variable)

public com.bea.language.codegen.IRParenthesizedExpression createParenthesizedExpression(
IRExpression expr)

public com.bea.language.codegen.IRReturnStatement createReturnStatement(
IRExpression expr)

public com.bea.language.codegen.IRStatementExpression createStatementExpression(
IRExpression expr)

public com.bea.language.codegen.IRSwitchBlockStatementGroup createSwitchBlockStatementGroup(
IRSwitchLabel label,
IRStatement statements,
IRSwitchBlockStatementGroup next)

public com.bea.language.codegen.IRSwitchLabel createSwitchLabel(
IRExpression caseExpr,
IRSwitchLabel next)

public com.bea.language.codegen.IRSwitchStatement createSwtichStatement(
IRExpression switchExpr,
IRSwitchBlockStatementGroup groups)

public com.bea.language.codegen.IRSynchronizedStatement createSynchronizedStatement(
IRExpression expr,
IRBlock body)

public com.bea.language.codegen.IRTernaryExpression createTernaryExpression(
int opcode,
IRExpression first,
IRExpression second,
IRExpression third)

public com.bea.language.codegen.IRThisExpression createThisExpression(
IReferenceType outerType,
IRName className)
Creates a reference to 'this' optionally qualified by the given class name.

public com.bea.language.codegen.IRDeleteStatement createDeleteStatement(
IRExpression expr)

public com.bea.language.codegen.IRThrowStatement createThrowStatement(
IRExpression expr)

public com.bea.language.codegen.IRTryStatement createTryStatement(
IRBlock body,
IRCatches catchClause,
IRFinally fin)

public com.bea.language.codegen.IRType createType(
int typeCode,
IType info)

public com.bea.language.codegen.IRTypeExpression createTypeExpression(
IRType type)

public com.bea.language.codegen.IRUnaryExpression createUnaryExpression(
int opcode,
IRExpression first)

public com.bea.language.codegen.IRVariableInitializer createVariableInitializer(
boolean isArray,
IRExpression initExpr,
IRArrayInitializer arrayInit,
IRVariableInitializer next)

public com.bea.language.codegen.IRWhileStatement createWhileStatement(
IRExpression condition,
IRStatement statement)

public void setRange(
IRNode node,
IToken first,
IToken last)
Sets the token range for the given node. It is required that the given node have been created with this interface.

Package com.bea.language.enterprise

This package provides information about the enterprise context of the project. Its classes and interfaces are described below.

public interface **com.bea.language.enterprise.ILangEntContext**

Provides additional context for enterprise projects.

Methods

public boolean isEntProject()
Determines if this is an enterprise project.

public IEntProject.IWebAppProject getWebAppProject()
Returns the web app project, if appropriate, for the current project.

public IEntProject.IWebAppProject[] getWebAppProjects()
Returns all web app projects in the current enterprise application.

public class **com.bea.language.enterprise.EntContext**

Provides access to the enterprise context for this project.

Constructors **public EntContext()**

Methods

public static com.bea.language.enterprise.ILangEntContext get()
Returns the enterprise context of compilation.

Package com.bea.language.feature

This package defines how a language extension accesses information about certain features of the source code. Its classes and interfaces are described below.

public interface **com.bea.language.feature.IProjectFeature**

The interface all features must provide to the framework.

Methods	**public java.lang.Object clone()** Creates a copy of this feature. **public void setProject(IProject proj)** Sets project on which this feature is to act.

public interface **com.bea.language.feature.ILangTokenUpdateFeature** implements com.bea.language.feature.ILangFeature

Provided by languages that want to perform updating of their token sets to changes in real time.

Methods	**public boolean isStartToken(IToken start)** Token updating code will iterate backward from a change looking for a valid start token. **public void setStartToken(IToken start)** Sets the start token for subsequent scanning. **public void setContext(ILangTokenUpdateFeature.IContext context)** Provides additional context for this feature. **public boolean isMatchingState(int newType, int oldType)** Determines from new and old token types whether the scanners that produced the tokens were in a matching state. It is assumed that token types can be mapped back to scanner states. A stateless scanner, should simply return true always. **public boolean getNextToken()** Adds the next token in the stream via the context.

public static interface **com.bea.language.feature.ILangTokenUpdateFeature.IContext**

Callback interface to the update framework.

Methods

public int getStart()
Returns the inclusive start of the changed text.

public int getOldEnd()
Returns the exclusive end of the removed text.

public int getNewEnd()
Returns the exclusive end of the inserted text.

public int getSizeChange()
Returns the size of the inserted text minus the removed text.

public com.bea.language.ILangReader getReader()
Returns a reader for the stream being re-tokenized.

public com.bea.language.ILangCompiler getLangCompiler()
Returns the compiler used to compile this node

public void addNextToken(
int type,
int size)
Adds the next token found in the stream.

public interface **com.bea.language.feature.ILangTokenInfoFeature** implements com.bea.language.feature.ILangFeature

Methods

public java.lang.String getTokenName(
IToken token)
Returns a displayable name for the token.

public java.lang.String[] getTokenNames()
Returns all the token names.

public interface **com.bea.language.feature.ILangStructureFeature** implements com.bea.language.feature.ILangFeature

Methods

public IStructureNode createStructure(
IStructureBuilder builder)
Creates a structure tree using the specified builder.

public Range getRangeInSource(
IStructureNode node)
Returns the range of the specified node in the source code.

public Range getRangeInSource(
IStructureNode node,
StructureCode propCode)
Returns the range of the specified property in the source code.

public java.lang.Object getSourceValue(
IStructureNode node,
StructureCode propCode)

Returns the syntactic value of the specified property.

public void addChild(
IStructureNode parent,
IAuthorNode child)
Updates the source to add the given child to the parent.

public void removeChild(
IStructureNode child)
Updates the source to remove the given child.

public void replaceNode(
IStructureNode oldNode,
IAuthorNode newNode)
Updates the source to replace the given node.

public interface **com.bea.language.feature.ILangSourceInfoFeature** implements com.bea.language.feature.ILangFeature

Methods

public java.lang.Object getIdentifierInfo(
int offset)
Returns the object referenced by the identifier at the given offset or null if the information could not be produced.

public boolean isCompletionChar(
int offset,
char ch)
Determines whether the given character, just typed, should initiate an attempt at completion (after a pause).

public Range getCompletionRange(
int offset)
Returns the range in the text that holds the current value of the completion that has been chosen. This is what will be replaced when a new completion is chosen.

public java.util.List getCompletions(
int offset,
char ch)
Returns all of the possible objects whose names could be used in place of the identifier at the given offset (or that would be at the given offset). The return value will be null if the information could not be produced.

public java.util.List getEnclosingContext(
int offset)
Returns information about the enclosing context of the given offset.

public interface **com.bea.language.feature.ILangPropertyInfoFeature** implements com.bea.language.feature.ILangFeature

Methods

public IDescription getDescription(
IStructureNode node)
Returns the description associated with the given node

public java.util.Iterator getAllTags(
IStructureNode node)
Returns the list of possible annotation tags on the given node. The iterator elements must be of type ITag

public interface **com.bea.language.feature.ILangFeature**

The interface all language specific features provide.

public static interface **com.bea.language.feature.ILangFeature.IContext**

The context provided to all language specific features.

Methods

public IToken getToken(
int offset)
Returns the token at the specified offset.

public java.util.Iterator getTokens()
Returns an iterator over all the language tokens starting at the beginning of the file. This iterator MUST be released with a call to release .

public java.util.Iterator getTokens(
boolean forward)
Returns an iterator over all the language tokens in the requested direction. This iterator MUST be released with a call to release .

public java.util.Iterator getTokens(
int offset,
boolean forward)
Returns an iterator over the language tokens starting at the given offset and moving in the given direction. This iterator MUST be released with a call to release .

public java.util.Iterator getTokens(
int offset,
int length,
boolean forward)
Returns an iterator over the language tokens starting at the given offset and moving in the given direction that will stop after the given length. This iterator MUST be released with a call to release .

public void release(
Iterator iter)
Releases the lock on the tokens held by the given iterator.

public java.lang.Object getParseTree()
Returns the parse tree for this language node.

public com.bea.language.ILineColumn toLineColumn(
int offset)
Returns the line and column of the given offset.

public java.lang.String getText(
int start,
int end)

Gets the text within the given range.

public void replaceText(
String text,
int start,
int end)
Replaces the text in the specified range with the specified text.

public INamespace getNamespace(
String name)
Returns the namespace with the given unique name.

public com.bea.language.ILangCompiler getLangCompiler()
Returns the compiler for this file.

public IFileStore.IEntry getEntry()
Returns the file entry for this feature.

public IProject getProject()
Returns the project for this feature.

public static class **com.bea.language.feature.ILangFeature.Context**

A static context object used to set and retrieve the context from TLS.

Constructors **public ILangFeature.Context()**

Methods **public static com.bea.language.feature.ILangFeature.IContext get()**
Gets the context.

public static void set(
ILangFeature.IContext c)
Sets the context.

public interface **com.bea.language.feature.IFileFeature** implements com.bea.language.feature.IProjectFeature

Defines the shape of a feature that pertains to a particular file in the project. These features can maintain state and can lock the state of the file as well.

Methods **public void setContext(**
IFileFeature.IContext context)
Sets the context for this feature.

public static interface **com.bea.language.feature.IFileFeature.IContext**

Defines the context available to an in-use feature.

Methods **public INode getRootNode()**
Returns the root language for this file.

public INode getNode(
　　int offset)
Returns inner-most language at the specified offset.

public java.util.List getNodes(
　　int offset)
Returns the languges from outer to inner at the given offset.

public IToken getToken(
　　INode node,
　　int offset)
Returns the token at the specified offset in the node.

public java.util.Iterator getTokens(
　　INode node,
　　int offset,
　　boolean forward)
Returns an iterator over this node's tokens starting at the given offset and moving in the given direction. This iterator MUST be released with a call to release .

public java.util.Iterator getTokens(
　　INode node,
　　int offset,
　　int length,
　　boolean forward)
Returns an iterator over this node's tokens starting at the given offset and moving in the given direction that will stop after the given length. This iterator MUST be released with a call to release .

public void lockState(
　　int state)
Locks the file into a particular compilation state.

public void releaseState()
Releases the lock on the file.

Package com.bea.language.fileinfo

This package defines interfaces implemented by compilers that wish to access or manipulate file information. Its classes and interfaces are described below.

public interface **com.bea.language.fileinfo.ILangFileResolver**

Implemented by compilers that wish to expose their content in the file namespace.

Methods　　**public com.bea.language.fileinfo.ILangFileResolver.IContent getContent()**
Returns an object recording the content of this file.

public static interface **com.bea.language.fileinfo.ILangFileResolver.IContent**

Represents the content of the file. Implementations will probably want to store the content in only one form, but may wish to make it available in other forms uppon request.

Methods	**public java.lang.String[] getContentTypes()** Returns the types of contents supported by this file. **public java.lang.Object getContent(String type)** Returns the content in the given type or null if not supported. **public boolean equals(Object o)** Compares this content to the given one. This must be implemented for changes to the namespace to be correctly determined.

public interface **com.bea.language.fileinfo.ILangFileCompiler** implements com.bea.language.ILangCompiler

Implemented by compilers that wish to participate more fully in the file namespace.

Methods	**public com.bea.language.fileinfo.ILangFileResolver getFileResolver()** Returns a resolver to use for the file namespace shape of this file.

Package com.bea.language.info

This package defines mutable versions of the compiler info interfaces of `com.bea.compiler.info`. Its classes and interfaces are described below.

public class **com.bea.language.info.LangInfoFactory**

Provides access to the factory of info structores.

Constructors	**public LangInfoFactory()**
Methods	**public static com.bea.language.info.ILangInfoFactory get()** Returns the singleton instance of ILangInfoFactory . **public static void set(ILangInfoFactory factory)** Sets the singleton instance of ILangInfoFactory . This should only be called by the implementation of the interface.

public interface **com.bea.language.info.IScope**

Represents a set of defined entities in the source. Note that this allows only for resolution of simple names. Typically, a language will add its own algorithm for resolving multi-part names on top of the scope.

Methods	**public IPackage resolvePackage(String name)** Returns the package with the given name or null if none exists.

public IReferenceType resolveType(
  String name)
Returns the type with the given short name or null if none exists.

public com.bea.language.info.ILangVariable resolveVariable(
  String name,
  boolean onlyStatic)
Returns the type of the variable (field or whatever) with the given short name or null if none exists.

public IReferenceType resolveMethod(
  String name,
  boolean onlyStatic)
Returns the type containing a method with the given short name or null if none exists.

public interface **com.bea.language.info.IMethodLocation**

Represents the location in which the method with the given ID was looked for (taking into account the argument types).

Methods

public IReferenceType getType()
Returns the type that must have contained the method.

public boolean isOnlyStatic()
Determines whether only static methods could be referenced.

public interface **com.bea.language.info.ILangVariable**

Encapsulates information about a variable defined in the source. This variable can be a local variable, a method parameter, a field of a user type, and so on.

Methods

public boolean isConstant()
Determines whether this variable has a constant initializer.

public com.bea.language.info.IConstant getConstant()
Returns the constant initializer of this variable.

public interface **com.bea.language.info.ILangUserType** implements com.bea.language.info.ILangReferenceTypeMember

A mutable version of IReferenceType . Note that no mutation should ever be performed after the type is handed to the framework.

Methods

public void setInterface(
  boolean isInterface)
Sets whether this type is an interface.

public void setCanonicalName(
  boolean hasCanonicalName)
Sets whether this type has a canonial name.

public void setAbstract(
boolean isAbstract)
Sets whether this type is not implemented.

public void setStatic(
boolean isStatic)
Sets whether this type has an outer type at run-time.

public void setFinal(
boolean isFinal)
Sets whether this type can be subclassed.

public void setStrictFP(
boolean isStrictFP)
Sets whether this type uses strict floating point operations.

public void setSynthetic(
boolean isSynthetic)
Sets whether this type is not available at compile-time.

public void setName(
IName name)
Sets the name of this type to the value given. Note that this should never be called on a type that has already been returned from the language to the framework.

public void setOuterType(
IReferenceType outerType)
Sets the outer type of this member. Note that this is called automatically for member types found during the resolve phase. This method is only needed for synthetic inner classes found during type checking (e.g. local and anonymous classes in Java).

public interface **com.bea.language.info.ILangReferenceTypeMember** implements com.bea.language.info.ILangDeclaredEntity

A mutable version of IReferenceTypeMember .

Methods

public void setModifiers(
int mods)
Sets all of the modifiers in an bitfield.

public interface **com.bea.language.info.ILangMethod** implements com.bea.language.info.ILangReferenceTypeMember, com.bea.language.info.ILangCallable

A mutable version of IMethod .

Methods

public com.bea.language.info.ILangMethod.ISignature getKey()
Returns an object representing the signature of this type, which is used as the key for this method in the method members map.

public void setID(
String id)
Sets the ID of this member.

public void setReturnType(
IType returnType)
Sets the return type of this method.

public void setAbstract(
boolean isAbstract)
Sets whether this method is not implemented.

public void setStatic(
boolean isStatic)
Sets whether this is a class method.

public void setFinal(
boolean isFinal)
Sets whether this method can be overriden.

public void setSynchronized(
boolean isSynchronized)
Sets whether this method is automatically synchronized.

public void setNative(
boolean isNative)
Sets whether this method is implemented in native code.

public void setStrictFP(
boolean isStrictFP)
Sets whether this method uses strict floating point operations.

public static interface **com.bea.language.info.ILangMethod.ISignature**

Defines the shape by which methods are said to define the same method.

Methods

public java.lang.String getName()
Returns the name of the method.

public IType getType()
Returns the tuple type of the method arguments.

public interface **com.bea.language.info.ILangInfoFactory**

Defines the interface for creation of compiler info structures.

For each type of structure that can be created, this provides one "constructor" that takes all of the information at creation time and another that returns a structure with null/default fields that must be filled in afterward.

Methods

public IName createName(
String id,
IName qual)
Returns a name with the given id and qualifier (can be null).

public IName createName(
String qname)
Returns a name from the given qualified name (containing '.'s).

public com.bea.language.info.ILangArrayType createArray()
Returns an array type with default properties.

public com.bea.language.info.ILangArrayType createArray(
    IType elemType)
Returns an array type with the given element type.

public com.bea.language.info.ILangUserType createClass()
Returns a class type with default properties.

public com.bea.language.info.ILangUserType createClass(
    Object source,
    String id,
    boolean isCanonical,
    String pkg,
    boolean isSynthetic)
Returns a class reference type with the given properties.

public com.bea.language.info.ILangUserType createClass(
    Object source,
    String id,
    boolean isCanonical,
    String pkg,
    IAccess access,
    boolean isAbstract,
    boolean isStatic,
    boolean isFinal,
    boolean isStrictFP,
    boolean isSynthetic,
    IAnnotationSet annotations)
Returns a class reference type with the given properties.

public com.bea.language.info.ILangUserType createInterface(
    Object source,
    String id,
    boolean isCanonical,
    String pkg,
    IAccess access,
    boolean isStrictFP,
    boolean isSynthetic,
    IAnnotationSet annotations)
Returns an interface reference type with the given properties.

public com.bea.language.info.ILangField createField()
Returns a field with default properties.

public com.bea.language.info.ILangField createField(
    Object source,
    String id,
    IType type,
    IAccess access,
    boolean isStatic,
    boolean isFinal,
    boolean isTransient,
    boolean isVolatile,
    IAnnotationSet annotations)
Returns a field with the given properties.

public com.bea.language.info.ILangMethod createMethod()
Returns a method with default properties.

public com.bea.language.info.ILangMethod createMethod(
Object source,
String id,
IType returnType,
List arguments,
List throwTypes,
IAccess access,
boolean isAbstract,
boolean isStatic,
boolean isFinal,
boolean isSynchronized,
boolean isNative,
boolean isStrictFP,
IAnnotationSet annotations)
Returns a method with the given properties.

public com.bea.language.info.ILangConstructor createConstructor()
Returns a constructor with default properties.

public com.bea.language.info.ILangConstructor createConstructor(
Object source,
List arguments,
List throwTypes,
IAccess access,
IAnnotationSet annotations)
Returns a constructor with the given properties.

public com.bea.language.info.ILangArgument createArgument()
Returns an argument with default properties.

public com.bea.language.info.ILangArgument createArgument(
String id,
IType type)
Returns an argument with the given name and type.

public com.bea.language.info.ILangAnnotationSet createAnnotationSet()
Creates an annotation set with default properties.

public com.bea.language.info.ILangAnnotationSet createAnnotationSet(
String shortDesc,
String desc,
ILangAnnotation root)
Creates an annotation set with the given properties.

public com.bea.language.info.ILangAnnotation createAnnotation()
Returns an annotation with the default properties.

public com.bea.language.info.ILangAnnotation createAnnotation(
Object source,
String id,
Object value,
IAccess access,
ILangAnnotation parent,
ILangAnnotation firstChild,
ILangAnnotation nextSibling)
Returns an annotation with the specified properties.

public IType createTupleType(
IType[] types)
Returns a tuple type with the given component types that can be used to look up methods on IReferenceType .

public interface **com.bea.language.info.ILangField** implements com.bea.language.info.ILangReferenceTypeMember, com.bea.language.info.ILangVariable

A mutable version of IField .

Methods	
	public void setID(**String id)** Sets the ID of this member.
	public void setType(**IType type)** Sets the type of this field.
	public void setStatic(**boolean isStatic)** Sets whether this is a class field.
	public void setFinal(**boolean isFinal)** Sets whether this field can be modified.
	public void setTransient(**boolean isTransient)** Sets whether this field is serialized.
	public void setVolatile(**boolean isVolatile)** Sets whether this field can change value unexpectedly.
	public void setConstant(**IConstant constant)** Sets the constant initializer value of this field.

public interface **com.bea.language.info.ILangDeclaredEntity**

A mutable version of IReferenceTypeMember .

Methods	
	public java.lang.Object getSource() Returns the node in the parse tree from which this was created.
	public void setSource(**Object source)** Sets the node in the parse tree from which this was created.
	public void setAccess(**IAccess access)** Records the accessability of this entity.
	public void setAnnotations(**IAnnotationSet annotations)** Records the annotations on this entity.

public interface **com.bea.language.info.ILangConstructor** implements com.bea.language.info.ILangDeclaredEntity, com.bea.language.info.ILangCallable

A mutable version of IConstructor .

public interface **com.bea.language.info.ILangCallable**

A mutable version of ICallable .

Methods	**public void setAccess(IAccess access)** Sets the accessability of this type. **public void setExceptionTypes(List types)** Sets the types that can be thrown by this method. **public void setArguments(List args)** Sets the arguments of this method.

public interface **com.bea.language.info.ILangArrayType** implements com.bea.language.info.ILangReferenceTypeMember

Represents a mutable version of array reference types.

Methods	**public void setElementType(IType elemType)** Sets the element type of the array.

public interface **com.bea.language.info.ILangArgument**

A mutable version of IArgument .

Methods	**public void setName(String name)** Returns the name of this argument. **public void setType(IType type)** Returns the type of this argument.

public interface **com.bea.language.info.ILangAnnotationSet**

Represents a mutable version of the annotation set, which also exposes the fact that the annotations are stored in a tree. Note that the root node of the tree is insignificant.

Methods

public void setShortDescription(
    String shortDesc)
Sets the short (one sentence) description of this entity.

public void setDescription(
    String desc)
Sets the full description of this entity.

public com.bea.language.info.ILangAnnotation getRoot()
Returns the root node of the annotation tree.

public void setRoot(
    ILangAnnotation root)
Records the root node of the annotation tree.

public interface **com.bea.language.info.ILangAnnotation**

Represents a mutable version of annotations. This version also exposes the fact that the annotations are arranged into a tree, which has links between siblings and from the parent to the first child.

Methods

public java.lang.Object getSource()
Returns the node in the parse tree from which this was created.

public void setSource(
    Object source)
Sets the node in the parse tree from which this was created.

public void setID(
    String id)
Records the ID of the annotation.

public void setAccess(
    IAccess access)
Records the accessability range of this annotation.

public void setValue(
    Object value)
Records the value of the annotation.

public com.bea.language.info.ILangAnnotation getParent()
Returns the parent of this annotation (if it has one).

public void setParent(
    ILangAnnotation parent)
Records the parent of this annotation (if it has one).

public com.bea.language.info.ILangAnnotation getFirstChild()
Returns the first child of this annotation.

public void setFirstChild(
    ILangAnnotation firstChild)
Records the first child of this annotation.

public com.bea.language.info.ILangAnnotation getNextSibling()

Returns the next sibiling of this annotation.

public void setNextSibling(
ILangAnnotation nextSibling)
Records the next sibiling of this annotation.

public interface **com.bea.language.info.ILangAmbiguous**

Represents a member of a reference type that is ambiguous. For fields and types, this means that the type inheritted two entities with the same name. For methods, this means that the type inheritted two methods with the same signature. In the case of methods, this is not ever an error.

Methods

public java.util.Iterator getMembers()
Returns the members that created the ambiguity.

public interface **com.bea.language.info.IConstantFactory**

Provides methods for creating implementations of IConstant .

Methods

public com.bea.language.info.IConstant create(
boolean val)

public com.bea.language.info.IConstant create(
char val)

public com.bea.language.info.IConstant create(
int val)

public com.bea.language.info.IConstant create(
long val)

public com.bea.language.info.IConstant create(
float val)

public com.bea.language.info.IConstant create(
double val)

public com.bea.language.info.IConstant create(
String val)

public interface **com.bea.language.info.IConstant**

Records the constant value of a field or expression in the source. This interface allows implementations to compute the values on demand. This can improve performance as computing the value of some constants (strings) can be very expensive.

Methods

public java.lang.Object getValue()
Returns the value of this constant, which will be an instance of Boolean , Character , Number , or String .

public com.bea.language.info.IConstant cast(

int type)
Casts this constant into one for the given primitive type.

public com.bea.language.info.IConstant convertToString()
Casts this constant into one for a string with this value.

public boolean fits(
int type)
Determines whether this primitive integral value will fit in the given primitive integral type.

public com.bea.language.info.IConstant operator(
int opcode)
Performs the given unary operation on this constant.

public com.bea.language.info.IConstant operator(
int opcode,
IConstant arg)
Performs the given binary operation on this and the given constant, which must be of the same type.

public com.bea.language.info.IConstant operator(
int opcode,
IConstant arg1,
IConstant arg2)
Performs the given ternary operation on this and the given constants. The two given arguments must be of the same type.

public class **com.bea.language.info.ConstantFactory**

Provides access to the singleton constant factory.

Constructors **public ConstantFactory()**

Methods **public static com.bea.language.info.IConstantFactory get()**
Returns the singleton constant factory.

public static void set(
IConstantFactory factory)
Sets the singleton constant factory. This should only be called by the implementation of IConstantFactory .

Package com.bea.language.java

This package defines how the Java language implementation can be extended to create support for other languages based on Java and its annotations. Its classes and interfaces are described below.

public class **com.bea.language.java.TransformSymbol**

Constructors **public TransformSymbol()**

Fields	**public static final SHORT**
	public static final IDENTIFIER
	public static final ANDEQ
	public static final GT
	public static final IMPLEMENTS
	public static final CONST
	public static final STRICTFP
	public static final NOTEQ
	public static final PLUSEQ
	public static final RBRACK
	public static final CATCH
	public static final COMMA
	public static final RBRACE
	public static final THROW
	public static final RPAREN
	public static final LBRACK
	public static final LT
	public static final ANDAND
	public static final OROR
	public static final DOUBLE
	public static final SUB
	public static final LBRACE
	public static final TRANSIENT
	public static final LPAREN
	public static final XOREQ
	public static final PROTECTED
	public static final INTEGER_LITERAL
	public static final NOT
	public static final COMMENT_SHORT
	public static final FINAL
	public static final FLOAT

```
public static final GOTO

public static final URSHIFTEQ

public static final PACKAGE

public static final JAVADOC_COMMENT

public static final COMP

public static final EQ

public static final BOOLEAN_LITERAL

public static final MOD

public static final CLASS

public static final SUPER

public static final ABSTRACT

public static final NATIVE

public static final LONG

public static final PLUS

public static final QUESTION

public static final WHILE

public static final EXTENDS

public static final INTERFACE

public static final CHAR

public static final BOOLEAN

public static final SWITCH

public static final DO

public static final FOR

public static final RSHIFTEQ

public static final VOID

public static final DIV

public static final PUBLIC

public static final RETURN

public static final MULT

public static final ELSE

public static final TRY
public static final GTEQ

public static final BREAK

public static final DOT

public static final INT

public static final NULL_LITERAL

public static final THROWS

public static final STRING_LITERAL

public static final EQEQ

public static final EOF

public static final SEMICOLON

public static final THIS

public static final DEFAULT

public static final MULTEQ

public static final IMPORT

public static final MINUS

public static final LTEQ

public static final OR

public static final error

public static final URSHIFT

public static final SYNCHRONIZED

public static final DIVEQ

public static final WHITESPACE

public static final LSHIFTEQ

public static final FINALLY

public static final CONTINUE

public static final INSTANCEOF

public static final IF

public static final MODEQ

public static final MINUSMINUS

public static final ASSERT

public static final COLON
```

public static final CHARACTER_LITERAL

public static final OREQ

public static final VOLATILE

public static final CASE

public static final PLUSPLUS

public static final NEW

public static final RSHIFT

public static final BYTE

public static final AND

public static final PRIVATE

public static final STATIC

public static final LSHIFT

public static final XOR

public static final COMMENT_LONG

public static final FLOATING_POINT_LITERAL

public static final MINUSEQ

public class **com.bea.language.java.JavaLanguageFactory**

Contains the reference to the template instance of the Java language.

Constructors	**public JavaLanguageFactory()**
Methods	**public static com.bea.language.java.IJavaLanguageFactory get()** Returns a reference to the implemention of IJavaLanguageFactory . **public static void set(** **IJavaLanguageFactory instance)** Sets the reference to the implementation of IJavaLanguageFactory . This will be called once when the implementation class is loaded. This should not be called but by the framework.

public class **com.bea.language.java.JavaFeatureContext**

Provides access to the implementation of IJavaFeatureContext .

Constructors	**public JavaFeatureContext()**
Methods	**public static com.bea.language.java.IJavaFeatureContext get()**

public static void set(
IJavaFeatureContext context)

public class **com.bea.language.java.JavaContext**

Provides access to the implementation of IJavaContext .

Constructors	**public JavaContext()**
Methods	**public static com.bea.language.java.IJavaContext get()** Returns the context for the Java compilation currently in progress.
	public static void set(**IJavaContext c)** Sets the context for the Java compilation currently in progress. This should only ever be called by the implementor of IJavaContext (i.e., the Java language).

public interface **com.bea.language.java.IJavaStructureNode**

Methods	**public StructureCode getTypeCode()**
	public void setTypeCode(**StructureCode typeCode)**
	public StructureCode getBaseTypeCode()

public static interface **com.bea.language.java.IJavaStructureNode.IUnit** implements com.bea.language.java.IJavaStructureNode

Methods	**public com.bea.language.java.IJavaStructureNode.IPackage getPackage()**
	public void setPackage(**IJavaStructureNode.IPackage pkg)**
	public java.util.List getImports()
	public void setImports(**List imports)**
	public java.util.List getClasses()
	public void setClasses(**List classes)**

public static interface **com.bea.language.java.IJavaStructureNode.IPackage**
implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getName()

public void setName(
        String name)
```

public static interface **com.bea.language.java.IJavaStructureNode.IImport**
implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getName()

public void setName(
        String name)

public boolean isMultiple()

public void setMultiple(
        boolean multiple)
```

public static interface **com.bea.language.java.IJavaStructureNode.IClass**
implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getName()

public void setName(
        String name)

public boolean isInterface()

public void setInterface(
        boolean interfaze)

public com.bea.language.java.IJavaStructureNode.IModifiers getModifiers()

public void setModifiers(
        IJavaStructureNode.IModifiers modifiers)

public java.util.List getDeclarations()

public void setDeclarations(
        List declarations)

public com.bea.language.java.IJavaStructureNode.IComment getComment()

public void setComment(
        IJavaStructureNode.IComment comment)
```

public static interface **com.bea.language.java.IJavaStructureNode.IMethod** implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getName()

public void setName(
        String name)

public java.lang.String getType()

public void setType(
        String type)

public com.bea.language.java.IJavaStructureNode.IModifiers getModifiers()

public void setModifiers(
        IJavaStructureNode.IModifiers modifiers)

public java.util.List getParameters()

public void setParameters(
        List parameters)

public com.bea.language.java.IJavaStructureNode.IComment getComment()

public void setComment(
        IJavaStructureNode.IComment comment)
```

public static interface **com.bea.language.java.IJavaStructureNode.IField** implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getName()

public void setName(
        String name)

public java.lang.String getType()

public void setType(
        String type)

public com.bea.language.java.IJavaStructureNode.IModifiers getModifiers()

public void setModifiers(
        IJavaStructureNode.IModifiers modifiers)

public com.bea.language.java.IJavaStructureNode.IComment getComment()

public void setComment(
        IJavaStructureNode.IComment comment)
```

public static interface **com.bea.language.java.IJavaStructureNode.IParameter** implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getName()

public void setName(
        String name)

public java.lang.String getType()

public void setType(
        String type)

public com.bea.language.java.IJavaStructureNode.IModifiers getModifiers()

public void setModifiers(
        IJavaStructureNode.IModifiers modifiers)
```

public static interface **com.bea.language.java.IJavaStructureNode.IModifiers** implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getAccess()

public void setAccess(
        String access)

public boolean isStatic()

public void setStatic(
        boolean _static)

public boolean isAbstract()

public void setAbstract(
        boolean _abstract)

public boolean isFinal()

public void setFinal(
        boolean _final)

public boolean isNative()

public void setNative(
        boolean _native)

public boolean isSynchronized()

public void setSynchronized(
        boolean _synchronized)

public boolean isTransient()

public void setTransient(
                boolean _transient)

public boolean isVolatile()

public void setVolatile(
        boolean _volatile)

public boolean isStrictFP()

public void setStrictFP(
        boolean _strictfp)
```

public static interface **com.bea.language.java.IJavaStructureNode.IComment** implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.util.List getTags()

public void setTags(
        List tags)
```

public static interface **com.bea.language.java.IJavaStructureNode.ITag** implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getName()

public void setName(
        String name)

public java.util.List getAttributes()

public void setAttributes(
        List attributes)
```

public static interface **com.bea.language.java.IJavaStructureNode.IAttribute** implements com.bea.language.java.IJavaStructureNode

Methods

```
public java.lang.String getName()

public void setName(
        String name)

public java.lang.String getValue()

public void setValue(
        String value)
```

public interface **com.bea.language.java.IJavaResolver**

Allows an extender of the Java language to change the compile-time visible shape of the class. To change the run-time shape of the class, implement IJavaGenerator .

Objects are created by the ILangInfoFactory returned by LangInfoFactory.get() .

Note that while all the methods below are defined in terms of adding new entities, extensions can also modify the entries already present. This should be done with great care, however, as such an operation has a high potential for confusing the (user) programmer.

Methods

public void addTypes(
List types)
Adds top-level types to the file. Each entry must be an instance of ILangUserType .

public void updateSuperTypes(
ILangUserType outerType,
ILangResolver.ITypeResolver.SuperTypes superTypes)
Allows the extension to modify the superclass and/or interfaces of the given outer type.

public void addMemberTypes(
ILangUserType outerType,
Map types)
Adds member types to a type defined in the file. The given map is from string IDs to ILangUserType s.

public void addMemberFields(
ILangUserType outerType,
Map fields)
Adds member fields to a type defined in the file. The given map is from string IDs to ILangField s.

public void addMemberMethods(
ILangUserType outerType,
Map methods)
Adds member methods to a type defined in the file. The given map is from ILangMethod.ISignature s to ILangMethod s.

public void addConstructors(
ILangUserType outerType,
Map constructors)
Adds constructors to a type defined in the file. The given map is from IType (tuples of arguments) to ILangConstructor s.

public void addAnnotations(
ILangUserType outerType,
ILangAnnotation parent,
List children)
Adds additional children to an annotation defined in the file. If the name of the parent is ROOT_NAME, then that would be adding new tags. Otherwise, it would be adding new attributes. Each entry in the given list must be an instance of ILangAnnotation .

public interface **com.bea.language.java.IJavaLanguageFactory**

Defines the shape of the Java language object, which can be cloned in order to create languages based on Java with annotations.

Methods **public com.bea.language.ILangLanguage createInstance(**
IJavaExtension extension)
Returns a new instance of the Java language with the given extensions.

public interface **com.bea.language.java.IJavaGenerator**

Allows an extender of the Java language to change the run-time shape of the class. To change the compile-time shape of the class, implement IJavaResolver .

Objects are created by the IIRFactory returned by IRFactory.get() .

Note that while all the methods below are defined in terms of adding new entities, extensions can also modify the entries already present. This should be done with great care, however, as such an operation has a high potential for confusing the (user) programmer.

Methods **public void addTypes(**
List types)
Adds top-level types to the file. Each entry must be an instance of IRClassDeclaration .

public void addBodyDeclarations(
IReferenceType outerType,
List decls)
Adds new members (types, fields, or methods), constructors, or initializer blocks to the body of the given class. Each entry must be an instanceof IRClassBodyDeclaration .

public void addAnnotations(
IDeclaredEntity entity,
IAnnotation parent,
List children)
Adds additional children to an annotation defined in the file. Each entry in the given list must be an instance of IRAnnotation .

public void addAnnotationAttributes(
IDeclaredEntity entity,
IAnnotation annotation,
List attributes)
Adds additional attributes to an annotation defined in the file. Each entry in the given list must be an instance of IRAnnotationAttribute .

public interface **com.bea.language.java.IJavaFeatureContext**

Provides additional context to a Java feature.

Methods **public com.bea.language.java.IJavaExtension getExtension()**
Returns the Java extension used to compile this file.

public interface **com.bea.language.java.IJavaFeature** implements com.bea.language.feature.ILangFeature

Defines extra context available to an extend feature of Java.

Methods **public void setContext(**
IJavaFeature.IContext context)
Sets the extra context available to Java features.

public static interface **com.bea.language.java.IJavaFeature.IContext**

Defines the extra context available to this feature.

Methods **public IDeclaredEntity getEntity(**
IStructureFeature.IStructureNode node)
Returns the info shape for the entity in the given node.

public interface **com.bea.language.java.IJavaExtension**

Defines the way to create extended versions of the Java language.

Methods **public java.lang.Object clone()**
Returns a new copy of this extension that is ready for use on a particular file.

public com.bea.language.java.IJavaResolver getResolver()
Returns the extensions to the resolve phase of Java compilation.

public com.bea.language.java.IJavaChecker getChecker()
Returns the extension for the checking phase of Java compilation.

public com.bea.language.java.IJavaGenerator getGenerator()
Returns the extensions to the codegen phase of Java compilation.

public com.bea.language.java.IJavaFeature getFeature(
Class clazz)
Returns an object that implements the given feature or null if the feature is not supported.

public com.bea.language.java.IJavaBuilder getBuilder()
Returns the builder for the structure tree of the file.

public interface **com.bea.language.java.IJavaContext**

Provides additional context to a Java extension.

Methods **public com.bea.language.ILangLocator getLocator()**
Returns a locator that can provide text ranges for errors found by the framework.

public com.bea.language.java.IJavaExtension getExtension()
Returns the Java extension currently compiling.

public com.bea.language.java.IJavaChecker getChecker()
Returns the Java extension currently checking.

public IDeclaredEntity getEntity()
Returns the entity whose annotations are currently being checked.

public IReferenceType resolveUserType(
String type,
IToken first,
IToken last)
Returns the user defined type with the given name or null if none exists in this scope.

public void addImport(
IRName name,
boolean isMultiple)
Don't use this method.

public interface **com.bea.language.java.IJavaChecker**

Allows a Java extension to perform further checks on the Java code. Diagnostics can be issued by calling Context.get().addDiagnostic(...) .

The container will make a depth-first traversal of the entities in the file, giving this extension checker a chance to perform checking on each of the entities found. During this process, the checker will be notified when the traversal is entering a particular type. Just for cleaniness, the container will check a type before it is entered (in theory, any such checking could be done in either startType or endType).

Each declared entity given will be either an IReferenceType , IField , IMethod , or IConstructor . It is not necessary to use instanceof , however, because IDeclaredEntity has methods that can determine what type of entity you have.

Methods

public void startFile(
IToken first,
IToken last)
Performs checking before any types are examined.

public com.bea.language.schema.IGrammar getAnnotationGrammar(
IDeclaredEntity entity,
IToken first,
IToken last)
Returns the grammar for the annotations above the given entity.

public void check(
IDeclaredEntity entity,
IToken first,
IToken last)
Performs additional checking on the given entity.

public void startType(
IReferenceType type,
IToken first,
IToken last)
Notifies the checker that we are about to begin checking the entities in the given type.

public void endType(
IReferenceType type,
IToken first,
IToken last)
Notifies the checker that we have finished checking the entities in the given type.

public void endFile(
IToken first,
IToken last)

Performs checking after all types are examined.

public interface **com.bea.language.java.IJavaBuilder**

Methods

public com.bea.language.java.IJavaStructureNode rebuildStructure(
  IJavaStructureNode node)
Takes a proto-Java structure tree and rebuilds it into the proper shape for the language extension.
Parameters
  node - the root of the tree
Returns
  the new tree

public int mapTypeCode(
  int code)
Maps type codes for this unit into type codes for Java.
Parameters
  code - the type code for the language extension
Returns
  the type code for the Java language

Package com.bea.language.schema

This package provides interfaces for building parsers and validity checkers for tree shaped data. Its classes and interfaces are described below.

public class **com.bea.language.schema.SchemaFactory**

Contains a reference to the singleton instance of the implementation of the ISchemaFactory interface.

Constructors

public SchemaFactory()

Methods

public static com.bea.language.schema.ISchemaFactory get()
Returns the singleton instance of ISchemaFactory .

public static void set(
  ISchemaFactory factory)
Sets the singleton instance of ISchemaFactory . This should be called only by the framework.

public interface **com.bea.language.schema.ISimpleType** implements com.bea.language.schema.IContentType

Represents a content type that is "simple" in the sense that its value is not parsed beyond tokenization on whitespace. Simple types are further categorized by whether they take their value as an atomic unit (after whitespace is transformed) or as a list of whitespace separated tokens.

Methods

public boolean isAtomic()

Determines whether this type takes its value as a single unit.

public boolean isList()
Determines whether this type takes its value as a list of tokens.

public interface **com.bea.language.schema.ISimpleOrderedType** implements com.bea.language.schema.ISimpleAtomicType

Represents a simple atomic type whose value set can be ordered.

Methods

public boolean isDecimal()
Determines whether this is a decimal type.

public boolean isDateTime()
Determines whether this is a date-time type.

public int compare(
    Object value1,
    Object value2)
Compares the given processed values and returns -1 for less than, 0 for equal, and 1 for greater than.

public interface **com.bea.language.schema.ISimpleListType** implements com.bea.language.schema.ISimpleType

Represents a simple type that takes its value as a whitespace separated list of tokens.

Methods

public java.util.List compile(
    Collection tokens)
Compiles the given value, which has been tokenized by whitespace as would StringTokenizer, into an intermediate form. If the value is found to be invalid, the checker should emit an error into the compilation context and return null.

Each token is an instance of the IListToken class below. These can be created by the schema factory.

public static interface **com.bea.language.schema.ISimpleListType.IListToken**

Represents a single token in the list to be compiled.

Methods

public java.lang.String getValue()
Returns the text of this token.

public IToken getFirst()
Returns the first underlying token that makes up this token.

public IToken getLast()
Returns the last underlying token that makes up this token.

public interface **com.bea.language.schema.ISimpleDecimalType** implements com.bea.language.schema.ISimpleOrderedType

Represents an ordered atomic type for decimal numbers.

Methods

public int getTotalDigits(
Object value)
Returns the total number of digits in the given compiled value.

public int getFractionDigits(
Object value)
Returns the total number of fractional digits in the compiled value.

public interface **com.bea.language.schema.ISimpleDateTimeType** implements com.bea.language.schema.ISimpleOrderedType

Represents an ordered atomic type for dates and times.

public interface **com.bea.language.schema.ISimpleAtomicType** implements com.bea.language.schema.ISimpleType

Represents a simple type that takes its value as a single atomic unit, after whitespace has been transformed.

Methods

public boolean isOrdered()
Determines whether this type is ordered.

public int getWhitespaceType()
Returns the operation to perform on the whitespace before validity checking a value of this type.

public java.lang.Object compile(
String value,
IToken first,
IToken last)
Compiles the given value into an intermediate form after transforming the whitespace as per the whitespace type. If the value is found to be invalid, the checker should emit an error into the compilation context and return null.

The returned value may be used for additional checking such as range checks or comparing to the fixed value.

public java.lang.Object compileTransformed(
String value,
IToken first,
IToken last)
Like compile , but assumes that the value has already been transformed for whitespace. This is intended for use by types that aggregate access to another type. Clients should normally call compile instead.

Fields

public static final WHITESPACE_PRESERVE

Indicates that whitespace should not be modified.

public static final WHITESPACE_REPLACE
Indicates that all whitespace should be transformed into space characters.

public static final WHITESPACE_COLLAPSE
Indicates that leading and trailing whitespace should be removed and each inner whitespace sequence should be transformed into a single space character.

public interface **com.bea.language.schema.ISchemaFactory**

Contains methods for creating schemas, types, and related objects. Built-in types and grammars are provided to handle (essentially) all of XML Schema and DTD. Note: collections passed in to create*() methods are not cloned, so it is caller's responsibility to maintain them.

Methods

public com.bea.language.schema.IQualifiedName createQualifiedName(
String qualifier,
String name)
Returns a qualified name with the given parts. The qualifier should be null if no qualifier was present.

public com.bea.language.schema.IElementSchema createElementSchema(
IQualifiedName name,
IElementType type,
Object defValue,
boolean fixed)
Returns an element schema with the given parts. Note that the default value must match the compilation type of the element.

public com.bea.language.schema.IElementType createElementType(
IAttributeGrammar attrGrammar,
IGrammar bodyGrammar)
Returns an element type with the given parts.

public com.bea.language.schema.IAttributeSchema createAttributeSchema(
IQualifiedName name,
IContentType type,
Object defValue,
boolean fixed)
Returns an attribute schema with the given parts. Note that the default value must match the compilation type of the attribute.

public com.bea.language.schema.IAttributeGrammar createEmptyAttributeGrammar()
Returns a grammar that accepts no attributes.

public com.bea.language.schema.IAttributeGrammar createAnyAttributeGrammar()
Returns a grammar that accepts any attributes.

public com.bea.language.schema.IAttributeGrammar createAttributeGrammar(
Collection attrOccurs)
Returns a grammar for the attributes with the given specifications. Each object in the collection must be an instance of IAttributeOccurance .

This grammar will verify that each attribute appears the number of times described in the specification. Note that more complex grammars could be created, but this interface does not provide that.

public com.bea.language.schema.ISchemaFactory.IAttributeOccurance

createAttributeOccurance(
IAttributeSchema attrSchema,
int useType)
Returns an attribute occurance with the given parts.

public com.bea.language.schema.IGrammar createEmptyGrammar()
Returns a grammar for a body that can contain no elements or non-whitespace content.

public com.bea.language.schema.IGrammar createAnyGrammar()
Returns a grammar that accepts any content and elements.

public com.bea.language.schema.IGrammar createSimpleGrammar(
IContentType type)
Returns a grammar for a body that can contain only content of the given type (i.e., no elements).

public com.bea.language.schema.IGrammar createComplexGrammar(
ISchemaFactory.IGroupOccurance occurs,
boolean mixed)
Returns a grammar for the group of elements with the given specification. The mixed parameter determines whether non-whitespace content is allowed between the elements.

Note that an "all" group is allowed only as the top-most group in the specification.

public com.bea.language.schema.ISchemaFactory.IElementOccurance createElementOccurance(
IElementSchema element,
int minOccurs,
int maxOccurs)
Returns an element occurance with the given parts.

public com.bea.language.schema.ISchemaFactory.IGroupOccurance createSequenceOccurance(
Collection members,
int minOccurs,
int maxOccurs)
Returns a sequence occurance with the given parts.

public com.bea.language.schema.ISchemaFactory.IGroupOccurance createChoiceOccurance(
Collection members,
int minOccurs,
int maxOccurs)
Returns a choice occurance with the given parts.

public com.bea.language.schema.ISchemaFactory.IGroupOccurance createAllOccurance(
Collection members,
int minOccurs,
int maxOccurs)
Returns an all occurance with the given parts. Note that the members must be element occurances whose max occurs is not greater than one. Futhermore, this can only be the top-most group in a grammar.

public com.bea.language.schema.ILanguageType createLanguageType(
IQualifiedName name,
ILangLanguage language)
Returns a language type for the given language.

public com.bea.language.schema.ISimpleDecimalType createDecimalType(
int type)
Returns a type object for the decimal type with the given code.

public com.bea.language.schema.ISimpleDateTimeType createDateTimeType(
int type)
Returns a type object for the date-time type with the given code.

public com.bea.language.schema.ISimpleOrderedType createOrderedType(
int type)
Returns a type object for the ordered type with the given code.

public com.bea.language.schema.ISimpleAtomicType createAtomicType(
int type)
Returns a stype object for the atomic type with the given code.

public com.bea.language.schema.ISimpleType createAnySimpleType()
Returns a type object for the any simple type.

public com.bea.language.schema.ISimpleType createSimpleType(
int type)
Returns a type object for the given built-in simple type.

public com.bea.language.schema.ISimpleListType createListType(
IQualifiedName name,
ISimpleAtomicType type)
Returns a type object for a whitespace delimited list of values of the given type.

public com.bea.language.schema.ISimpleListType.IListToken createListToken(
String value,
IToken first,
IToken last)
Returns a list token that can be passed to the compile method of a list type.

public com.bea.language.schema.ISimpleType createUnionType(
IQualifiedName name,
Collection type)
Returns a type object whose value set is the union of the value sets of the given simple types.

public com.bea.language.schema.ISimpleAtomicType createPatternRestrictedType(
IQualifiedName name,
ISimpleAtomicType type,
String pattern,
boolean caseSensitive,
String message)
Returns a type object whose value set is the value set of the given type intersected with the given pattern.

public com.bea.language.schema.ISimpleAtomicType
createEnumerationRestructedType(
IQualifiedName name,
ISimpleAtomicType type,
Object[] values)
Returns a type object whose value set is the value set of the given type intersected with the given list of possible values. Each value must be of the compiled type of the underlying type.

public com.bea.language.schema.ISimpleType createLengthRestrictedType(
IQualifiedName name,
ISimpleType type,
int minLength,
int maxLength)
Returns a type object whose value set is the value set of the given type intersected with the values whose lengths are in the given range. Note that length means the number of characters for atomic types but the number of non-whitespace tokens for list types.

public com.bea.language.schema.ISimpleDecimalType createPrecisionRestrictedType(
IQualifiedName name,
ISimpleDecimalType type,
int maxTotalDigits,
int maxFractionDigits)
Returns a type object whose value set is the value set of the given decimal type restricted to the given numbers of total and fractional digits. Note that -1 can be passed to indicate an unbounded maximum.

public com.bea.language.schema.ISimpleOrderedType createRangeRestrictedType(
IQualifiedName name,
ISimpleOrderedType type,
Object minValue,
boolean minInclusive,
Object maxValue,
boolean maxInclusive)
Returns a type object whose value set is the value set of the given ordered type restricted to the given range of values. Either value can be null to indicate no bound on that side. Note that these values' types must match the compilation type of type .

public com.bea.language.schema.ISimpleAtomicType createWhitespaceRestrictedType(
IQualifiedName name,
ISimpleAtomicType type,
int wspaceType)
Returns a type object whose value set is the value set of the given atomic type with the whitespace transformation changed as indicated.

public java.lang.String transformWhitespace(
String value,
int wspaceType)
Performs the given whitespace transformation on the value.

Fields

public static final TYPE_DECIMAL
Represents the type of decimal numbers.

public static final TYPE_INTEGER
Represents the type of decimal numbers without fractions.

public static final TYPE_NONNEGATIVEINTEGER
Represents the type of integers that are not positive.

public static final TYPE_NONPOSITIVEINTEGER
Represents the type of integers that are not negative.

public static final TYPE_NEGATIVEINTEGER
Represents the type of integers that are negative.

public static final TYPE_POSITIVEINTEGER
Represents the type of integers that are not positive.

public static final TYPE_LONG
Represents the type of integers that are limited to 64 bits.

public static final TYPE_UNSIGNEDLONG
Represents the type of integers that are limited to 64 unsigned bits.

public static final TYPE_INT
Represents the type of integers that are limited to 32 bits.

public static final TYPE_UNSIGNEDINT
Represents the type of integers that are limited to 32 unsigned bits.

public static final TYPE_SHORT
Represents the type of integers that are limited to 16 bits.

public static final TYPE_UNSIGNEDSHORT
Represents the type of integers that are limited to 16 unsigned bits.

public static final TYPE_BYTE
Represents the type of integers that are limited to 8 bits.

public static final TYPE_UNSIGNEDBYTE
Represents the type of integers that are limited to 8 unsigned bits.

public static final TYPE_DATETIME
Represents the XML date-time type.

public static final TYPE_TIME
Represents the XML time type.

public static final TYPE_DATE
Represents the XML date type.

public static final TYPE_GYEARMONTH
Represents the XML year-month type.

public static final TYPE_GYEAR
Represents the XML year type.

public static final TYPE_GMONTHDAY
Represents the XML month-day type.

public static final TYPE_GDAY
Represents the XML day type.

public static final TYPE_GMONTH
Represents the XML month type.

public static final TYPE_FLOAT
Represents the IEEE single precision floating point type.

public static final TYPE_DOUBLE
Represents the IEEE double precision floating point type.

public static final TYPE_DURATION
Represents the XML Schema duration type.

public static final TYPE_STRING
Represents the type of strings with whitespace preserved.

public static final TYPE_NORMALIZEDSTRING
Represents the type of strings with whitespace replaced.

public static final TYPE_TOKEN
Represents the type of strings with whitespace collapsed.

public static final TYPE_LANGUAGENAME
Represents the type of identifiers for natural languages.

public static final TYPE_BOOLEAN
Represents the boolean type.

public static final TYPE_HEXBINARY

Represents the type of binary numbers encoded in hex.

public static final TYPE_BASE64BINARY
Represents the type of binary numbers encoded in base 64.

public static final TYPE_URI
Represents the type of all valid URIs.

public static final TYPE_XML_NMTOKEN
Represents the type of tokens whose characters can legally appear in an XML name.

public static final TYPE_XML_NAME
Represents the type of valid names in XML documents.

public static final TYPE_XML_NCNAME
Represents the type of valid XML names without colons.

public static final TYPE_XML_QNAME
Represents the type of qualified names, that is, a URI followed by a colon followed by an NCName.

public static final TYPE_JAVA_ID
Represents the type of valid Java identifiers.

public static final TYPE_JAVA_NAME
Represents the type of (optionally) qualified Java names.

public static final TYPE_ANY
Represents the any simple type.

public static final TYPE_XML_NMTOKENS
Represents a list of the NMToken type.

public static interface **com.bea.language.schema.ISchemaFactory.IAttributeOccurance**

Provides a specification for how an attribute is allowed to appear in the attribute list of a particular element.

Methods

public com.bea.language.schema.IAttributeSchema getAttribute()
Returns the schema for the attribute in question.

public int getUseType()
Returns the rule for how it can be used.

Fields

public static final USE_PROHIBITED
Indicates that the attribute cannot be used.

public static final USE_OPTIONAL
Indicates that the attribute can optionally be used.

public static final USE_REQUIRED
Indicates that the attribute must be used.

public static interface

com.bea.language.schema.ISchemaFactory.IOccurance

Provides a specification for how an element or group of elements is allowed to appear in the body of a particular element.

Methods	**public boolean isElement()** Determines whether this is an IElementOccurance .
	public boolean isGroup() Determines whether this is an IGroupOccurance .
	public int getMinOccurs() Returns the minimum number of times this group or element may occur. This must be less than or equal to max occurs.
	public int getMaxOccurs() Returns the maximum number of times this group or element may occur. This must be greater than or equal to min occurs.

public static interface **com.bea.language.schema.ISchemaFactory.IElementOccurance** implements com.bea.language.schema.ISchemaFactory.IOccurance

Provides a specification for how an element is allowed to appear in the body of a particular element.

Methods	**public com.bea.language.schema.IElementSchema getElement()** Returns the schema for the element in question.

public static interface **com.bea.language.schema.ISchemaFactory.IGroupOccurance** implements com.bea.language.schema.ISchemaFactory.IOccurance

Provides a specification for how a group is allowed to appear in the body of a particular element.

Methods	**public int getGroupType()** Returns the type (i.e. operation) of this group.
	public java.util.Collection getGroupMembers() Returns the members of this group (i.e. the groups or elements to which the group rule applies).
Fields	**public static final TYPE_SEQUENCE** Indicates that this group allows the members to appear in the order provided to it.
	public static final TYPE_CHOICE Indicates that this group allows one of the members to occur for each instance of the group.
	public static final TYPE_ALL Indicates that this group allows its members to occur in any order. Futhermore, the members must all be elements.

public interface **com.bea.language.schema.IQualifiedName**

Represents a namespace qualified name of an element in the tree.

Methods

public java.lang.String getQualifier()
Returns the qualifier on the name.

public java.lang.String getName()
Returns the name without the qualifier.

public interface **com.bea.language.schema.ILanguageType** implements com.bea.language.schema.IContentType

Represents a content type is parsed and checked as a nested language of the host language.

Methods

public com.bea.language.ILangLanguage getLanguage()
Returns the language object for this content type.

public interface **com.bea.language.schema.IGrammar**

Encapsulates the rules about how content and elements can appear in the source. Each instance of the grammar is intended to operate as a finite state machine. (Although it need not be implemented as one.)

The only restriction built into this interface is that it must be possible to determine the schema for the next element given only its name and the names of the previous elements. Hence, a finite state machine is almost always sufficient for describing the grammar.

Information about the sequence of names found is delivered in the requests that are made of the grammar. Whenever content is found, a call is made to getNextContent . Whenever an element is found, a call is made to getNextElement .

Methods

public java.lang.Object clone()
Returns a new instance of the state machined that is ready for use.

```
public boolean begin(
        IToken first,
        IToken last)
```
Notifies the machine that processing will now begin. The return value indicates whether no errors were found.

```
public com.bea.language.schema.IContentType getNextContent(
        IToken first,
        IToken last)
```
Returns the type for the content that appears next or null if it is illegal to have content here.

```
public com.bea.language.schema.IElementSchema getNextElement(
        IQualifiedName name,
        IToken first,
        IToken last)
```
Returns the schema for the element with the given name or null if this element does not match the grammar.

public boolean end(

IToken first,
IToken last)
Notifies the machine that all input has been seen. The return value indicates whether no errors were found.

public interface **com.bea.language.schema.IElementType**

Provides information about the legal content of an element.

Methods

public com.bea.language.schema.IAttributeGrammar getAttributeGrammar()
Returns a grammar for the attributes of this element.

public com.bea.language.schema.IGrammar getBodyGrammar()
Returns a grammar for the body of this element.

public interface **com.bea.language.schema.IElementSchema**

Provides information about the legal shape of an element in the tree.

Methods

public com.bea.language.schema.IQualifiedName getName()
Returns the qualified name of this element.

public com.bea.language.schema.IElementType getType()
Returns the type of this element.

public java.lang.Object getDefaultValue()
Returns the default value of this element or null if it has none. The default value's type must match the compilation type of the element.

public boolean isFixed()
Determines whether the value of this element must be equal to its default value.

public interface **com.bea.language.schema.IContentType**

Provides information about the legal values for "content" in the tree. Content appears as the value of an attribute or in the body of an element.

We separate content types into two varieties: simple and language. The former parses as a string of whitespace separated text and is checked for validity by calling a method. The latter uses its own parser and type checker, which can be completely generic.

Methods

public com.bea.language.schema.IQualifiedName getName()
Returns the name of this type.

public boolean isSimple()
Determines whether this is a simple content type.

public boolean isLanguage()
Determines whether this is a language content type.

public interface **com.bea.language.schema.IAttributeSchema**

Provides information about the legal shape of an attribute on an element in the tree.

Methods

public com.bea.language.schema.IQualifiedName getName()
Returns the qualified name of this attribute. Normally, the qualifier on the name will be null.

public com.bea.language.schema.IContentType getType()
Returns the type of this attribute.

public java.lang.Object getDefaultValue()
Returns the default value of this element or null if it has none. The default value's type must match the compiled type of the attribute.

public boolean isFixed()
Determines whether the value of this element must be equal to its default value.

public interface **com.bea.language.schema.IAttributeGrammar**

Encapsulates the rules about how attributes can appear on a particular element in the tree. As with IGrammar , this grammar is intended to operate as a finite state machine.

Methods

public java.lang.Object clone()
Returns a new instance of the state machined that is ready for use.

public boolean begin(
IToken first,
IToken last)
Notifies the machine that processing will now begin. The return value indicates whether no errors were found.

public com.bea.language.schema.IAttributeSchema getNextAttribute(
IQualifiedName name,
IToken first,
IToken last)
Returns the schema for the attribute that with the given name or null if this attribute does not match the grammar.

public boolean end(
IToken first,
IToken last)
Notifies the machine that all input has been seen. The return value indicates whether no errors were found.

Package com.bea.language.transform

This package provides interfaces for building transforming compilers that transform a source language to a target language. Its classes and interfaces are described below.

public class **com.bea.language.transform.TransformTargetScanner**

Exposes a base level ITransformTargetScanner implementation for use in implementing target language specific versions. This implementation maintains the list of tokens, and implements a simple fake token type for insertion among real tokens gathered from the source language parse.

Constructors	**public TransformTargetScanner()**
Methods	**public static void set(TransformTargetScanner.IBaseTransformTargetScanner base)** **public static com.bea.language.transform.ITransformTargetScanner createBaseScanner()**

public static interface **com.bea.language.transform.TransformTargetScanner.IBaseTransformTargetScanner** implements com.bea.language.transform.ITransformTargetScanner

Methods	**public java.lang.Object clone()** Create a new instance of this base target scanner.

public interface **com.bea.language.transform.ITransformTargetScanner**

An in memory scanner that allows one language to transform its parse tree into a list of tokens for parsing by a target language.

Methods

public void addToken(
IToken token)
Adds a token scanned from the document, already in the target language, presumably by way of a nested language in the original parse.

public void addToken(
int type,
String text)
Adds a fake token to the scanner.
Parameters
type - public token type exposed in the target language's public API
text - text string of the token, or null if no string applies

public void addToken(
int type,
String text,
IToken start,
IToken end)
Adds a fake token to the scanner, with a given range in the original file.
Parameters
type - public token type exposed in the target language's public API
text - text string of the token, or null if no string applies
start - starting token of the range (from source language parse)
end - ending token of the range (from source language parse)

public IToken getNextToken()
Returns the next token in the stream or null if the end has been reached. Each token returned will be either a normal token from the scripting scanner or a fake token produced by the

transform. The latter tokens will implement the ITransformToken interface. For these tokens, the provided text must be used instead of requesting the text from the file.

public static interface **com.bea.language.transform.ITransformTargetScanner.ITransformToken**

Extends IToken to provide text for tokens that did not come from the real source code.

Methods **public java.lang.String getText()**
Returns the text of this token.

public interface **com.bea.language.transform.ITransformSourceScanner**

Supports a simple scan of a section of the source file being transformed, returning language native tokens of the target language for use in the transform.

Methods **public IToken getNextToken()**
Returns the next token in the input stream or null if the end has been reached. This token should have been added to the token set.

public interface **com.bea.language.transform.ILangTransformParser** implements com.bea.language.ILangParser

A parser that allows its tokens to be supplied by a transform scanner. ILangTransformCompiler.getParser() returns an ILangParser that may be cast to ILangTransformParser.

Methods **public com.bea.language.transform.ITransformTargetScanner createTransformScanner()**
Creates a new transform scanner for this parser. When the parse() method on this parser is called, it will read tokens from the transform scanner.
Returns
a language specific transform scanner

public interface **com.bea.language.transform.ILangTransformLanguage** implements com.bea.language.ILangLanguage

Marker interface for a language that returns an ILangTransformCompiler from its getCompiler() method.

public interface **com.bea.language.transform.ILangTransformCompiler** implements com.bea.language.ILangCompiler

ILangTransformLanguage.getCompiler returns an ILangCompiler which may be cast to an

ILangTransformCompiler.

Methods

public com.bea.language.transform.ITransformSourceScanner getScanner(
ILangReader reader)
Returns a new file scanner for this language.

Package com.bea.language.typeinfo

This package provides interface for languages that need to access or manipulate type information . Its classes and interfaces are described below.

public interface **com.bea.language.typeinfo.ILangTypeNamespace**

Exposes additional methods available to languages.

Methods

public void setTypes(
List types)
Sets the types of the file being compiled.

public void resolve(
ILangResolver.ITypeResolver resolver,
IReferenceType refType)
Performs complete resolution of the given type using the given resolver. This is used for resolving types that are not visible in the namespace and have special context that is needed when resolving.

If this attempts to resolve a type that is currently resolving, then an exception will be thrown. This situation indicates that a cyclic dependency has been encountered in the files.

PHASE: resolve, check
Parameters
resolver - The resolver to use for resolving this type.
refType - The type to resolve.

public void addAllMembersDependency(
IReferenceType target)
Adds a dependency on all members of the given type.

public com.bea.language.info.ILangInfoFactory getFactory()
Returns the factory for type namespace objects.

public IType getType(
String signature)
Returns the type object corresponding to the given signature.

public boolean setExternalFile(
ILangUserType type,
String path)
Used to update the file for inner class .class files.

public IName getClassName(
IFile file)
Get the expected name of a class defined in the given file.

Package com.bea.language.xscript

This package provides interfaces for accessing and manipulating XScript code (JavaScript with native XML support). Its classes and interfaces are described below.

public class **com.bea.language.xscript.XScriptLanguageFactory**

Contains the reference to the template instance of the Java language.

Constructors	**public XScriptLanguageFactory()**
Methods	**public static com.bea.language.xscript.IXScriptLanguageFactory get()** Returns a reference to the implemention of IJavaLanguageFactory . **public static void set(IXScriptLanguageFactory instance)** Sets the reference to the implementation of IJavaLanguageFactory . This will be called once when the implementation class is loaded. This should not be called but by the framework.

public interface **com.bea.language.xscript.IXScriptLanguageFactory**

Defines the shape of the XScript language object, which can be cloned in order to create languages based on XScript with annotations.

Methods	**public com.bea.language.ILangLanguage createInstance(IJavaExtension extension)** Returns a new instance of the Java language with the given extensions.

public interface **com.bea.language.xscript.IXSCodeBlock**

Methods	**public void setName(String icbName)** **public java.lang.String getName()** **public void setFunctionType(byte icbFunctionType)** **public byte getFunctionType()** **public void setICode(byte[] icbICode)** **public byte[] getICode()** **public void setICodeTop(int icbICodeTop)**

```
public int getICodeTop()

public void setStringTable(
          String[] icbStringTable)

public java.lang.String[] getStringTable()

public void setStringTableIndex(
          int icbStringTableIndex)

public int getStringTableIndex()

public void setNumberTable(
          Number[] icbNumberTable)

public java.lang.Number[] getNumberTable()

public void setNumberTableIndex(
          int icbNumberTableIndex)

public int getNumberTableIndex()

public void setMaxStack(
          int icbMaxStack)

public int getMaxStack()

public void setMaxLocals(
          int icbMaxLocals)

public int getMaxLocals()

public void setMaxTryDepth(
          int icbMaxTryDepth)

public int getMaxTryDepth()

public com.bea.language.xscript.IXSCodeBlock[] createCodeBlocks(
          int entries)

public void setNestedFunctions(
          IXSCodeBlock[] icbNestedFunctions)

public com.bea.language.xscript.IXSCodeBlock[] getNestedFunctions()

public void setVariableTable(
          String[] icbVariableTable)

public java.lang.String[] getVariableTable()

public void setVariableIndex(
          int icbVariableIndex)

public int getVariableIndex()

public void setSourceFile(
          String icbSourceFile)

public java.lang.String getSourceFile()

public void setLineNumberTable(
```

Hashtable icbSLineNumberTable)

public java.util.Hashtable getLineNumberTable()

public void markAsValid()

public byte[] serialize()

public com.bea.language.xscript.IXSCodeBlock deserialize(
byte[] serializedData)

Fields

public static final FUNCTION_STATEMENT
There are three types of functions that can be defined. The first is a function statement. This is a function appearing as a top-level statement (i.e., not nested inside some other statement) in either a script or a function. The second is a function expression, which is a function appearing in an expression except for the third type, which is... The third type is a function expression where the expression is the top-level expression in an expression statement. The three types of functions have different treatment and must be distinquished.

public static final FUNCTION_EXPRESSION

public static final FUNCTION_EXPRESSION_STATEMENT

The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to one of ordinary skill in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.